Santander customer transaction prediction

Submitted by

SANTOSH KUMAR NAYAK

Santander customer transaction prediction

Contents

Chapter 1

Introduction

1.1 Problem Statement

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

1.2 Data

In this project, our task is to build classification models which would be used to predict which customers will make a specific transaction in the future. Given below is a sample of the Santander customer transaction dataset:

Table 1.1: Train dataset (Columns:1-202)

ID_code	target	var_0	var_1	var_2	………	……….	var_199
train_01	0	8.92	-6.78	11.90	……....	……..	-1.09
train_02	0	11.5	-4.14	13.85	……...	...……	1.95
train_03	0	8.60	-2.74	12.08	...……	...……	0.39
train_04	0	11.06	-2.15	8.95	...……	...……	-8.99
train_05	0	9.83	-1.48	12.87	...……	...……	-8.81

Table 1.2: Test Dataset (Columns: 1-201)

ID_code	var_0	var_1	var_2	var_3	………	……….	var_199
test_01	11.06	8.92	-6.78	11.90	…......	……..	-1.09
test_02	8.53	11.5	-4.14	13.85	……...	...…….	1.95
test_03	5.48	8.60	-2.74	12.08	...……	...…….	0.39
test_04	8.53	11.06	-2.15	8.95	...……	...…….	-8.99
test_05	11.7	9.83	-1.48	12.87	...…….	...…….	-8.81

From the table below we have the following 16 variables, using which we have to predict the bike rental count:

Table 1.3: Predictor Variables

| SL.No.	Predictor
1	ID-code
2	var0
3	var1
4	var2
5	var3
6	var4
7	var5
….	.. …
	…..
….	……
….	……
….	…….
….	…….
….	…….
….	……..
202	var199

Chapter 2

Methodology

2.1 Exploratory Data Analysis (EDA)

Exploratory data analysis is one of the most important step in data mining in order to know features of data. It involves the loading dataset, target classes count, data cleaning, typecasting of attributes, missing value analysis, Attributes distributions and trends. So, we have to clean the data otherwise it will effect on performance of the model. Now we are going to explain one by one as follows. In this EDA I explained with seaborn visualizations.

2.2.1 Target classes count
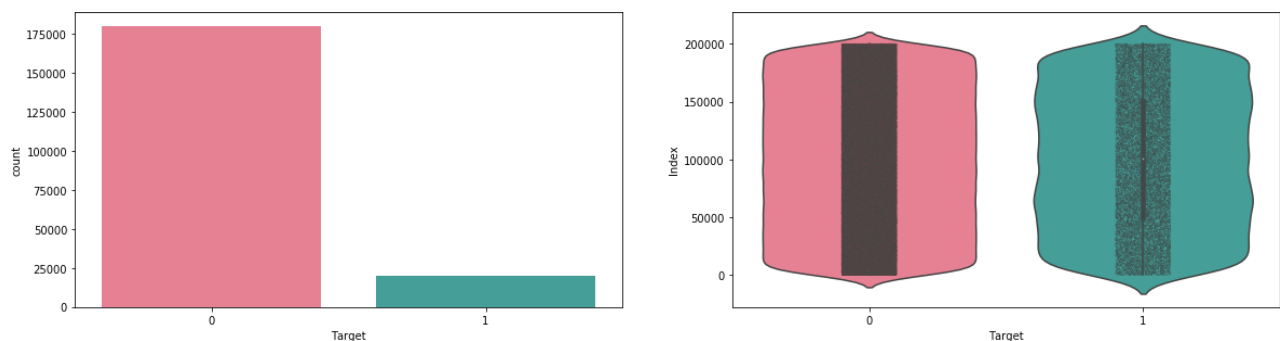
Take away:

- We have a unbalanced data, where 90% of the data is the number of customers those will not make a transaction and 10% of the data is those who will make a transaction.
- Look at the violin plots seems that there are no relationship between the targets with the index of the train data frame. This is more dominated by the zero targets then for the ones.
- Look at the jitter plots with violin plots. We can observed that targets looks uniformly distributed over the index of the data frame.

2.2.2 Missing value Analysis

In this, we have to find out any missing values are present in dataset. If it's present then either delete or impute the values using mean, median and KNN imputation method. We have not found any missing values in both train and test data.

R and Python code as follows,

```
#Missing values in train and test data
# R code
missing_val<-data.frame(missing_val=apply(train_df,2,function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
[1] 0
missing_val<-data.frame(missing_val=apply(test_df,2,function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
[1] 0

# Python code
train_missing=train_df.isnull().sum().sum()
[1] 0
test_missing=test_df.isnull().sum().sum()
[1] 0
```

2.1.1 Attributes distributions and trends

Distribution of train attributes

Let us look distribution of train attributes from var_0 to var_99
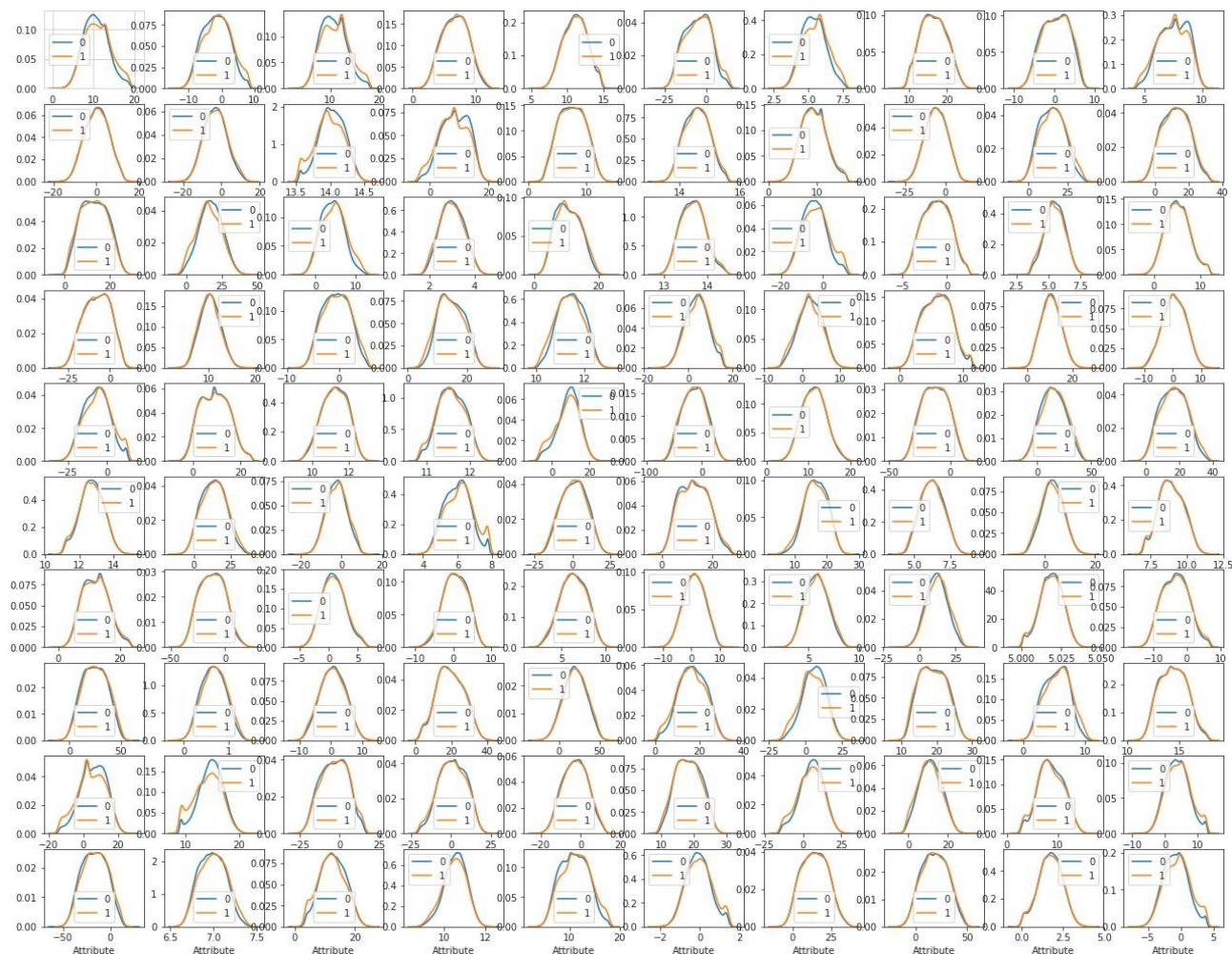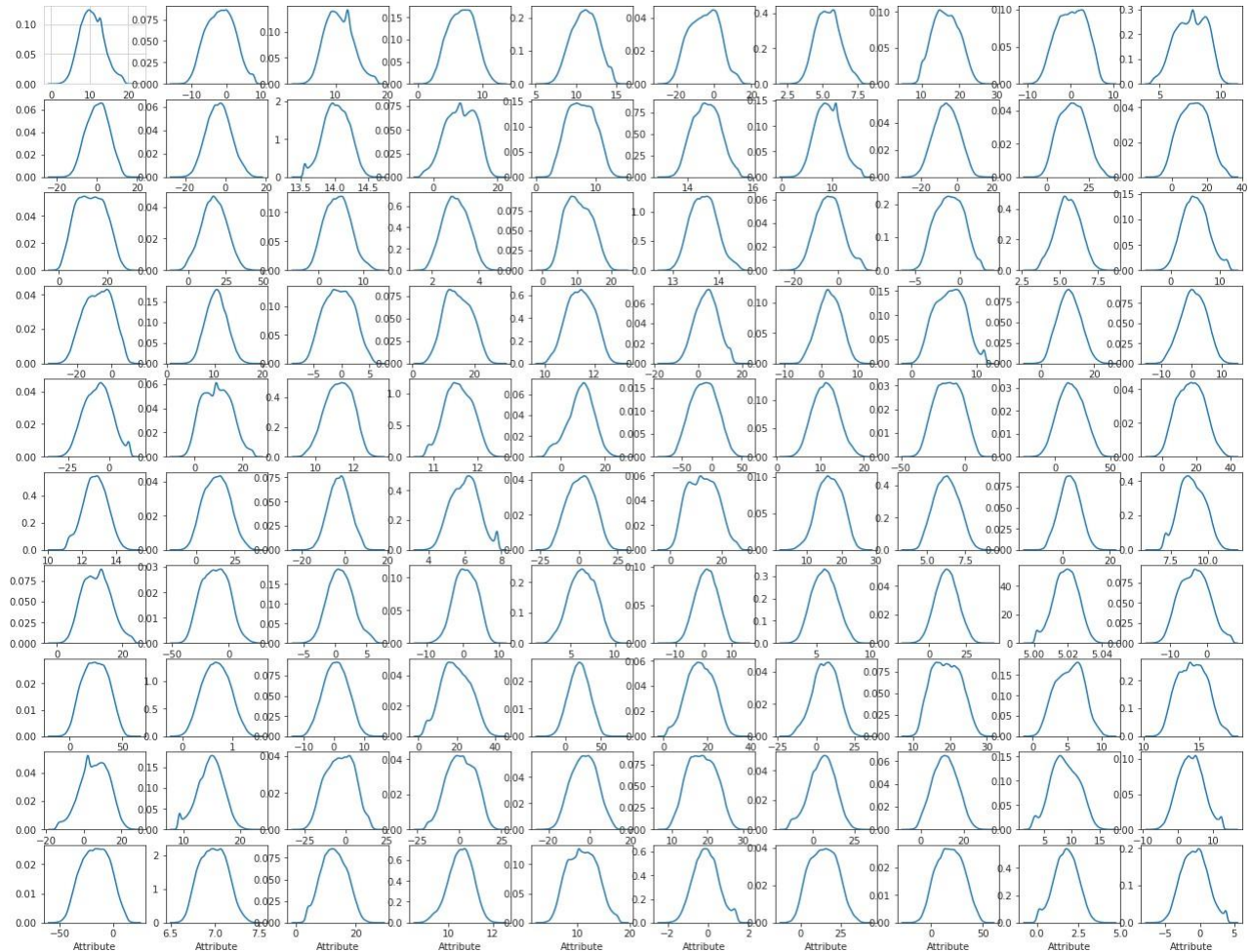

Take away:

- We can observed that there is a considerable number of features which are significantly have different distributions for two target variables. For example like var_0,var_1,var_9,var_19,var_18 etc.
- We can observed that there is a considerable number of features which are significantly have same distributions for two target variables. For example like var_3, var_7,var_10,var_17,var_35 etc.

Distribution of test attributes

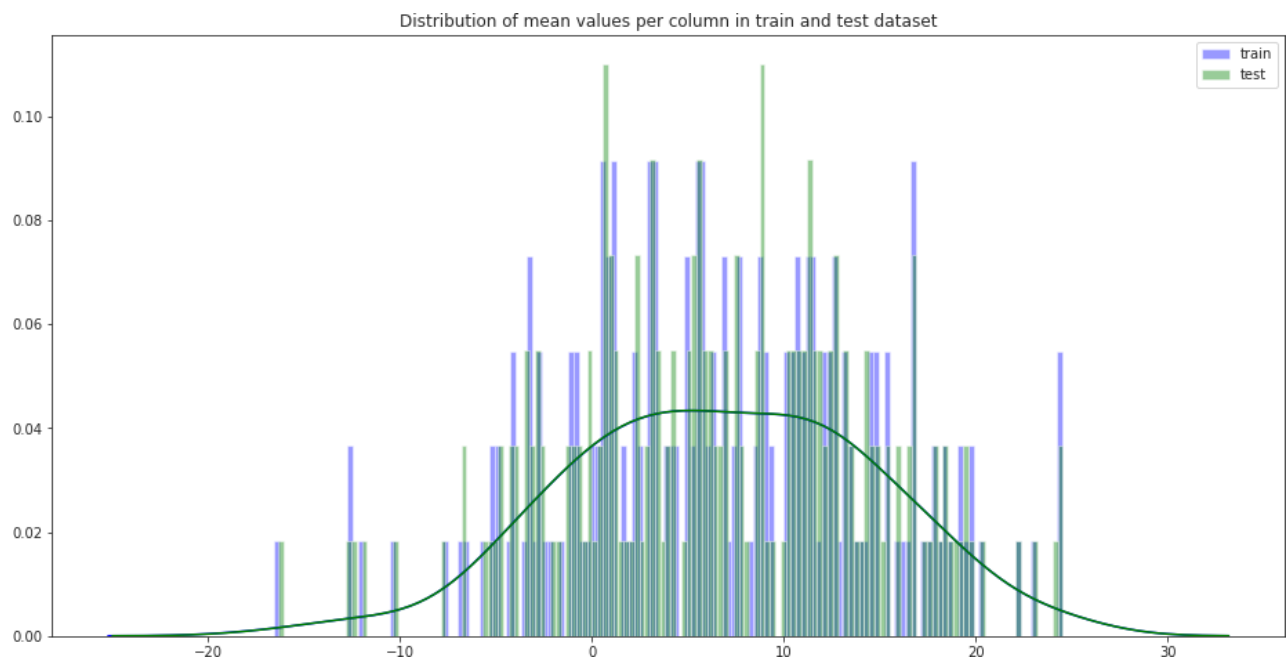Let us look distribution of test attributes from var_0 to var_99


Take away:

- We can observed that there is a considerable number of features which are significantly have different distributions. For example like var_0, var_1, var_9, var_18 var_38 etc.
- We can observed that there is a considerable number of features which are significantly have same distributions. For example like var_3, var_7, var_10, var_17, var_45, var_192 etc.

Distribution of mean values in both train and test dataset

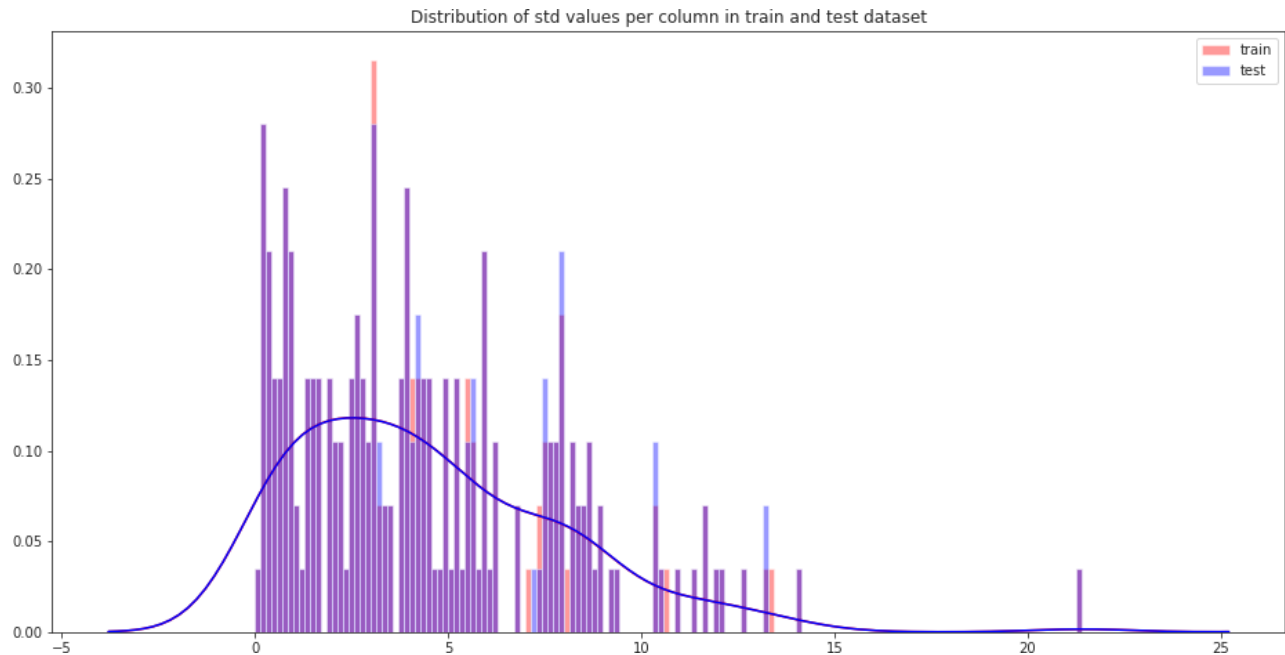Let us look distribution of mean values per column in train and test dataset

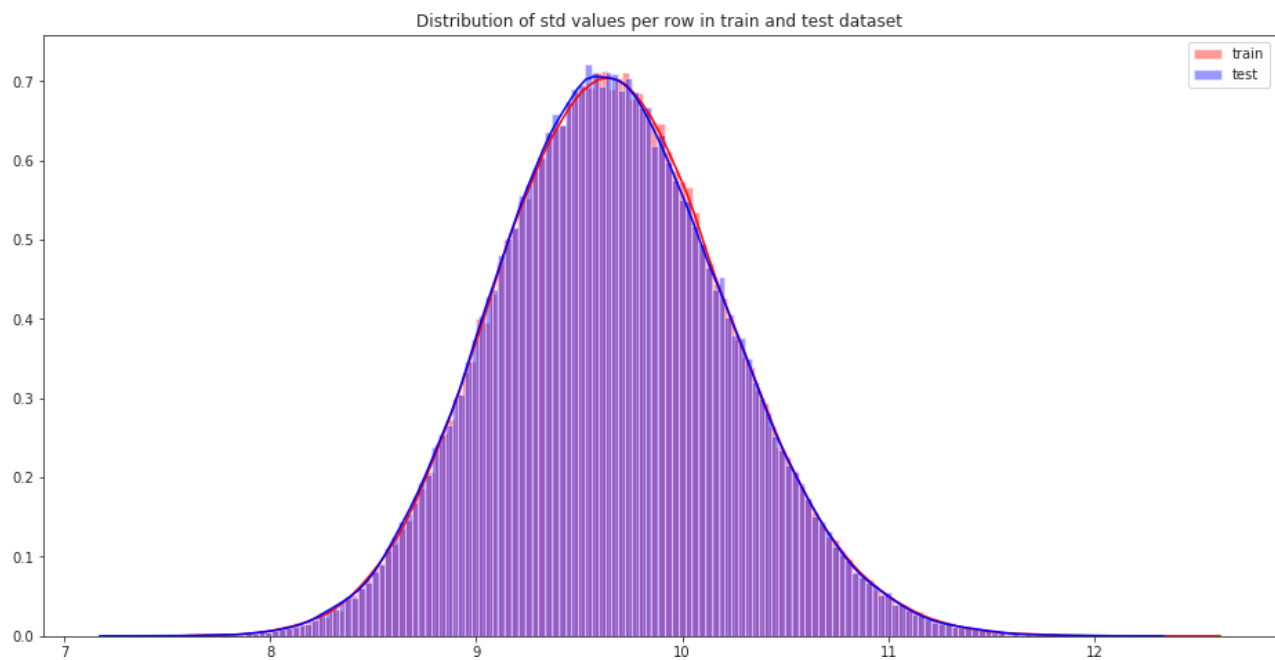Let us look distribution of mean values per row in train and test dataset
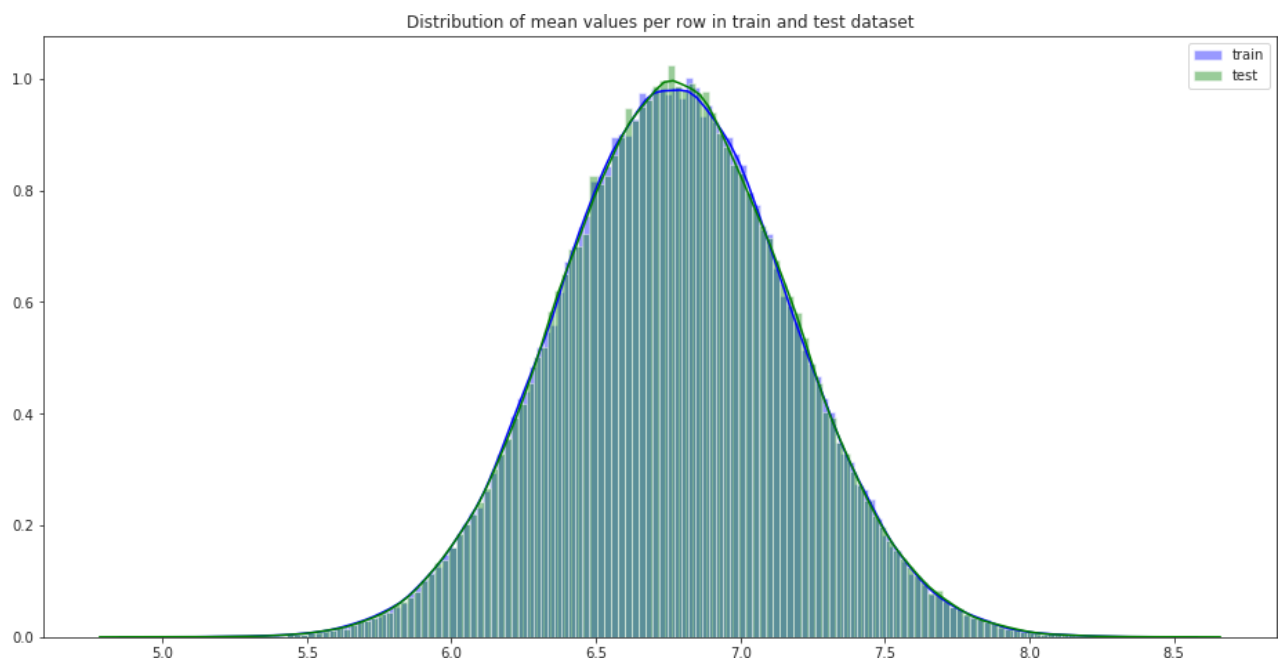
Distribution of standard deviation (std) values in train and test dataset

Let us look distribution of standard deviation (std) values per column in train and test dataset
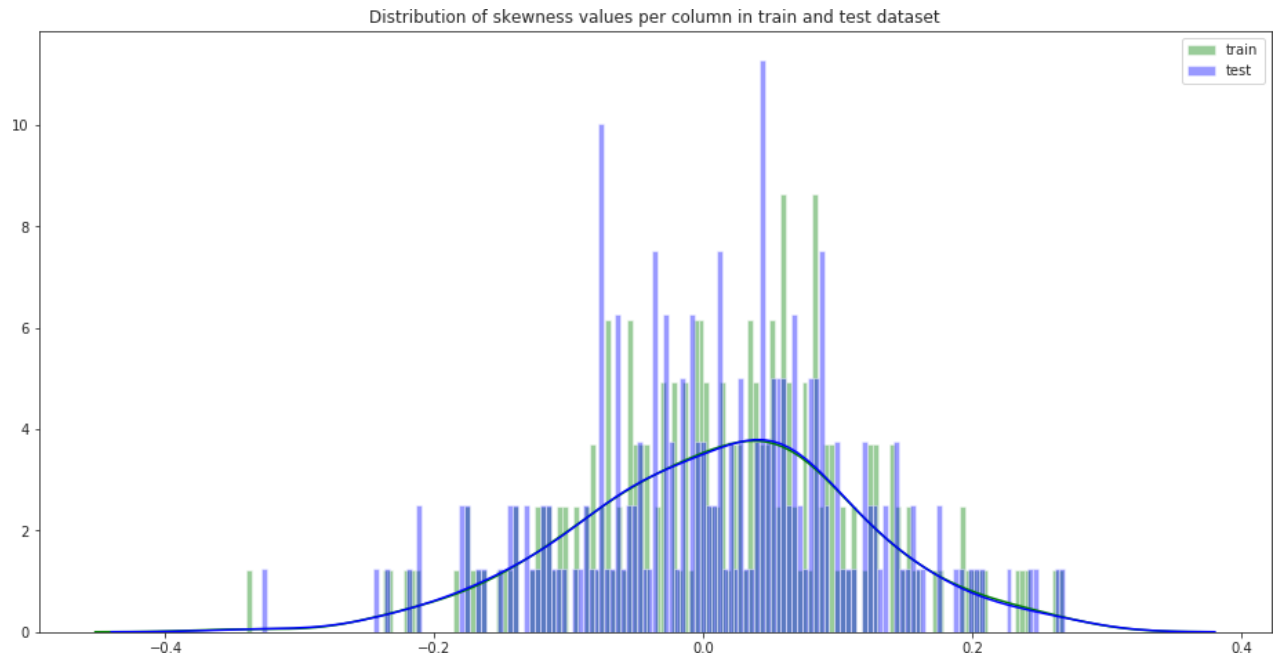

Distribution of std values per column in train and test dataset

Let us look distribution of standard deviation (std) values per row in train and test dataset

Distribution of std values per row in train and test dataset

Distribution of skewness values in train and test dataset

Let us look distribution of skewness values per column in train and test dataset

Distribution of skewness values per column in train and test dataset

Let us look distribution of skewness per row in train and test dataset

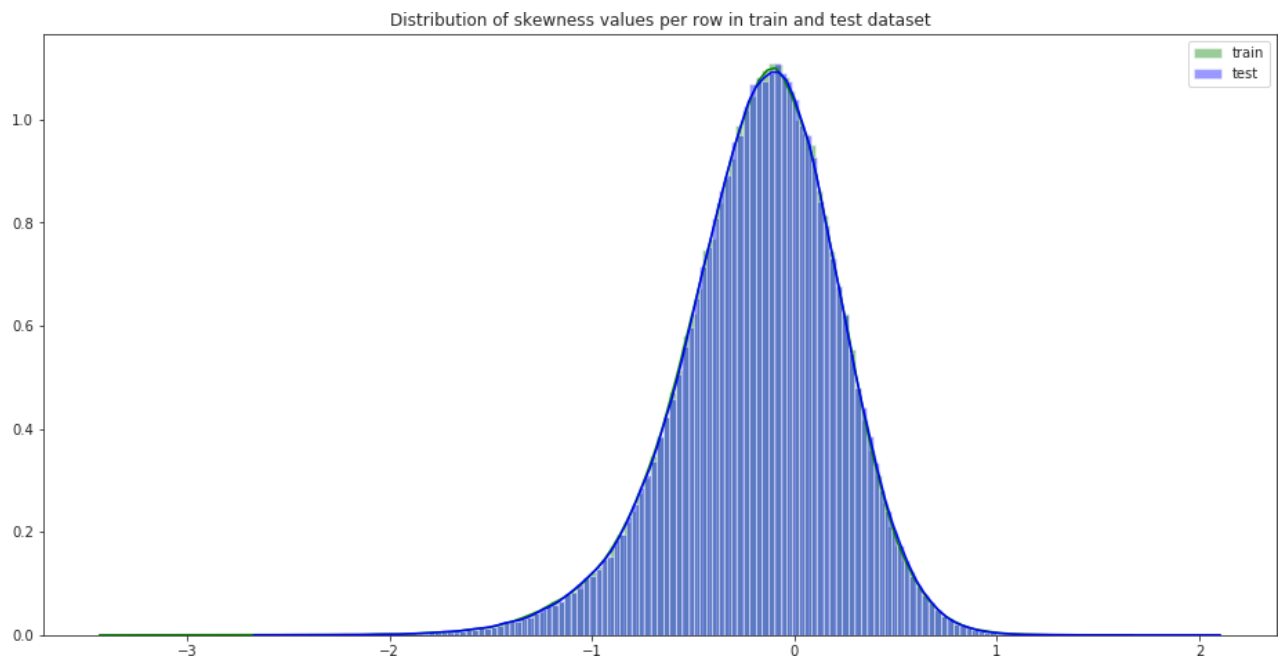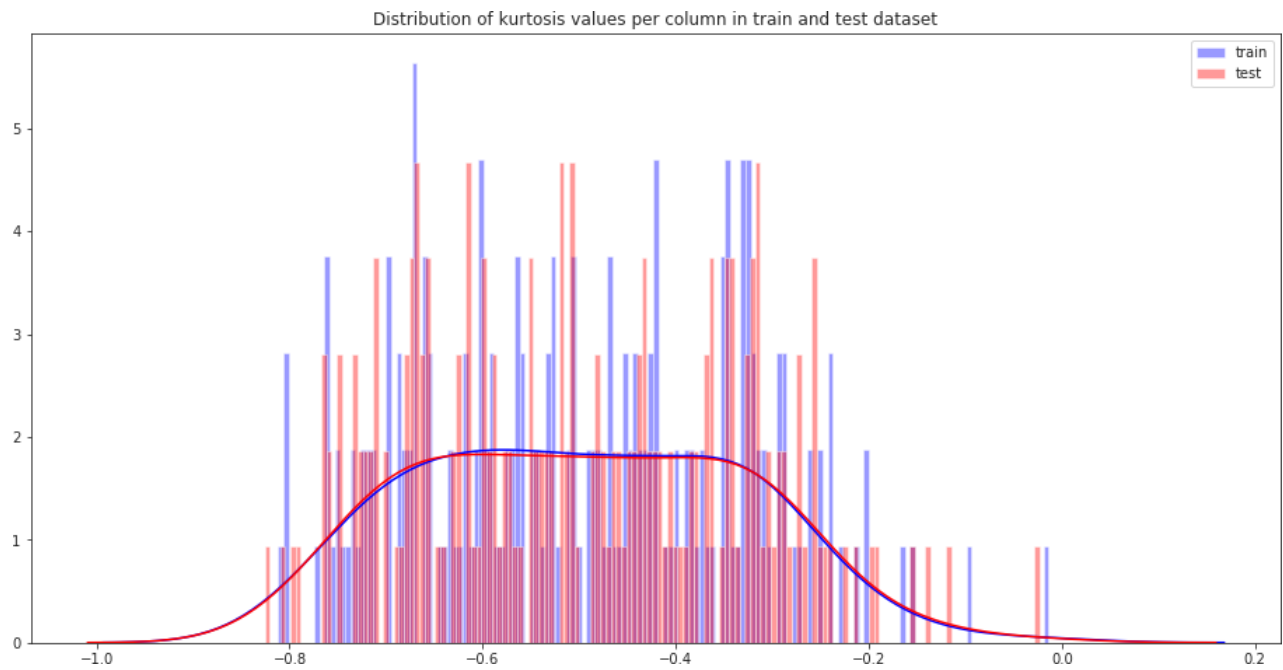Distribution of skewness values per row in train and test dataset
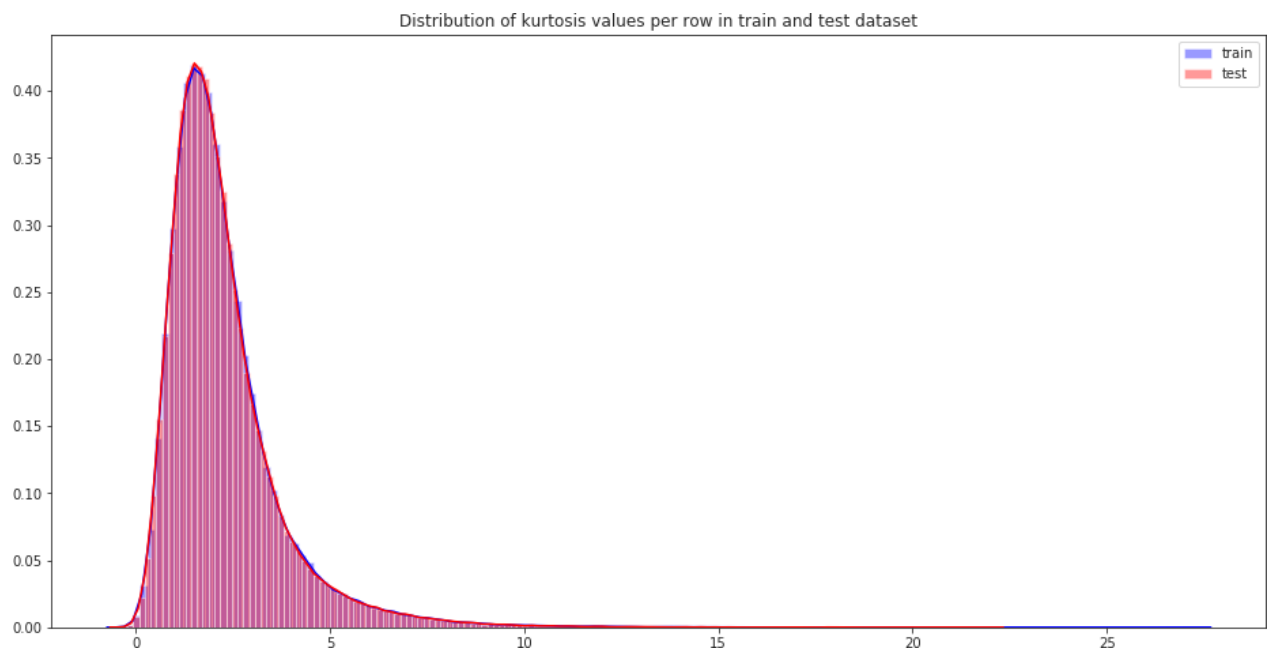
Distribution of kurtosis values in train and test dataset

Let us look distribution of kurtosis values per column in train and test dataset

Let us look distribution of kurtosis values per row in train and test dataset

2.1.2 Outlier analysis

In this project, we haven't perform outlier analysis due to the data is imbalanced and also not required for imbalanced data.

2.1.3 Feature Selection

Feature selection is very important for modelling the dataset. The every dataset have good and unwanted features. The unwanted features would effect on performance of model, so we have to delete those features. We have to select best features by using ANOVA, Chi-Square test and correlation matrix statistical techniques and so on. In this, we are selecting best features by using Correlation matrix.

Correlation matrix

Correlation matrix is tells about linear relationship between attributes and help us to build better models.

From correlation distribution plot, we can observed that correlation between both train and test attributes are very small. It means that all both train and test attributes are independent to each other.

Correlation distribution plot for train and test attributes
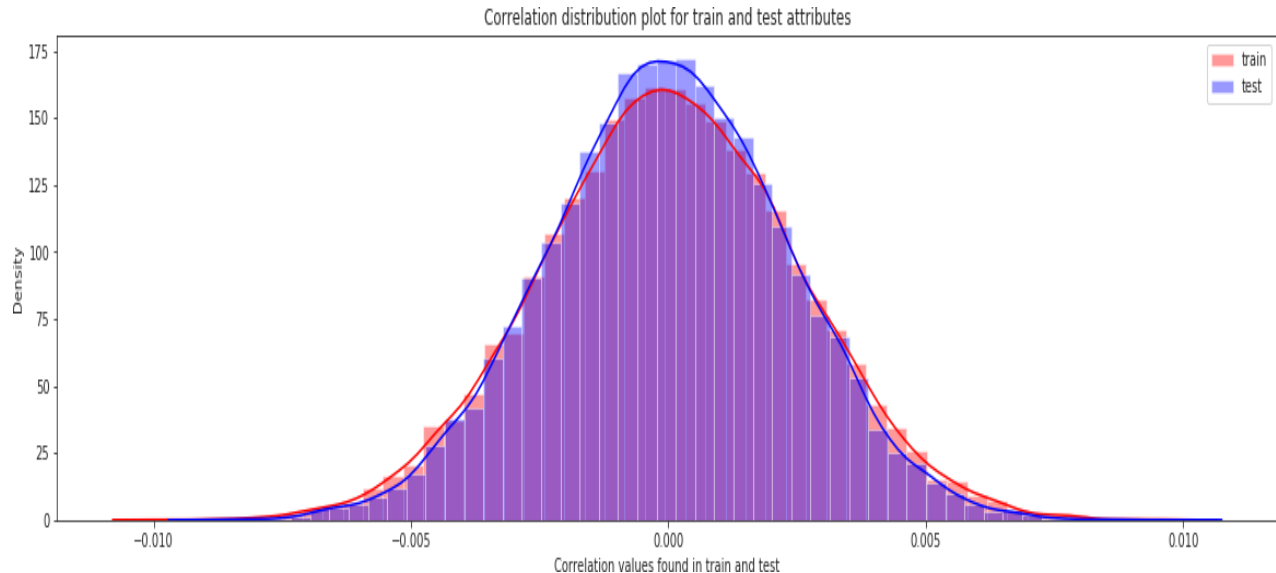
2.1.4 Feature engineering

Let us do some feature engineering by using

- Permutation importance
- Partial dependence plots

Permutation importance

Permutation variable importance measure in a random forest for classification and regression. The variables which are mostly contributed to predict the model.
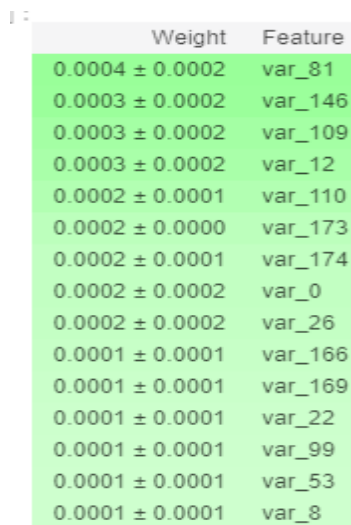
Python code

```
#training data
X=train_df.drop(columns=['ID_code','target'],axis=1)
test=test_df.drop(columns=['ID_code'],axis=1)
y=train_df['target']

#Split the training data
X_train,X_valid,y_train,y_valid=train_test_split(X,y,random_state=42)

#Random forest classifier
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)
#fitting the model
rf_model.fit(X_train,y_train)

#Let us calculate weights and show important features using eli5 library.
from eli5.sklearn import PermutationImportance
perm_imp=PermutationImportance(rf_model,random_state=42)
#fitting the model
perm_imp.fit(X_valid,y_valid)

#Important features
eli5.show_weights(perm_imp,feature_names=X_valid.columns.tolist(),top=200)
```

Weight	Feature
0.0004 ± 0.0002	var_81
0.0003 ± 0.0002	var_146
0.0003 ± 0.0002	var_109
0.0003 ± 0.0002	var_12
0.0002 ± 0.0001	var_110
0.0002 ± 0.0000	var_173
0.0002 ± 0.0001	var_174
0.0002 ± 0.0002	var_0
0.0002 ± 0.0002	var_26
0.0001 ± 0.0001	var_166
0.0001 ± 0.0001	var_169
0.0001 ± 0.0001	var_22
0.0001 ± 0.0001	var_99
0.0001 ± 0.0001	var_53
0.0001 ± 0.0001	var_8

R code

```
#Split the training data
train_index<-sample(1:nrow(train_df),0.75*nrow(train_df))
train_data<-train_df[train_index,]
valid_data<-train_df[-train_index,]

#Training the Random forest classifier
set.seed(2732)
train_data$target<-as.factor(train_data$target)
mtry<-floor(sqrt(200))
tuneGrid<-expand.grid(.mtry=mtry)
rf<-randomForest(target~.,train_data[,-c(1)],mtry=mtry,ntree=10,importance=TRUE)

#Variable importance
VarImp<-importance(rf,type=2)
VarImp
```

Variable importance based on Mean Decrease Gini

|	MeanDecreaseGini
var_0	178.19905
var_1	179.11005
var_2	186.08049
var_3	124.84417
var_4	115.25478
var_5	136.53634
var_6	193.64204
var_7	111.92811
var_8	109.93159
var_9	159.85075
var_10	117.02210

Take away:

- We can observed that the top important features are var_12, var_26, var_22, var_174, var_198 and so on based on Mean decrease Gini.

Partial dependence plots

Partial dependence plot gives a graphical depiction of the marginal effect of a variable on the class probability or classification. While feature importance shows what variables most affect predictions, but partial dependence plots show how a feature affects predictions.

Python code

```
#Create the data we will plot 'var_53'
 features=[v for v in X_valid.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature=
'var_53')
 #plot feature "var_53"
 pdp.pdp_plot(pdp_data,'var_53')
 plt.show()
```
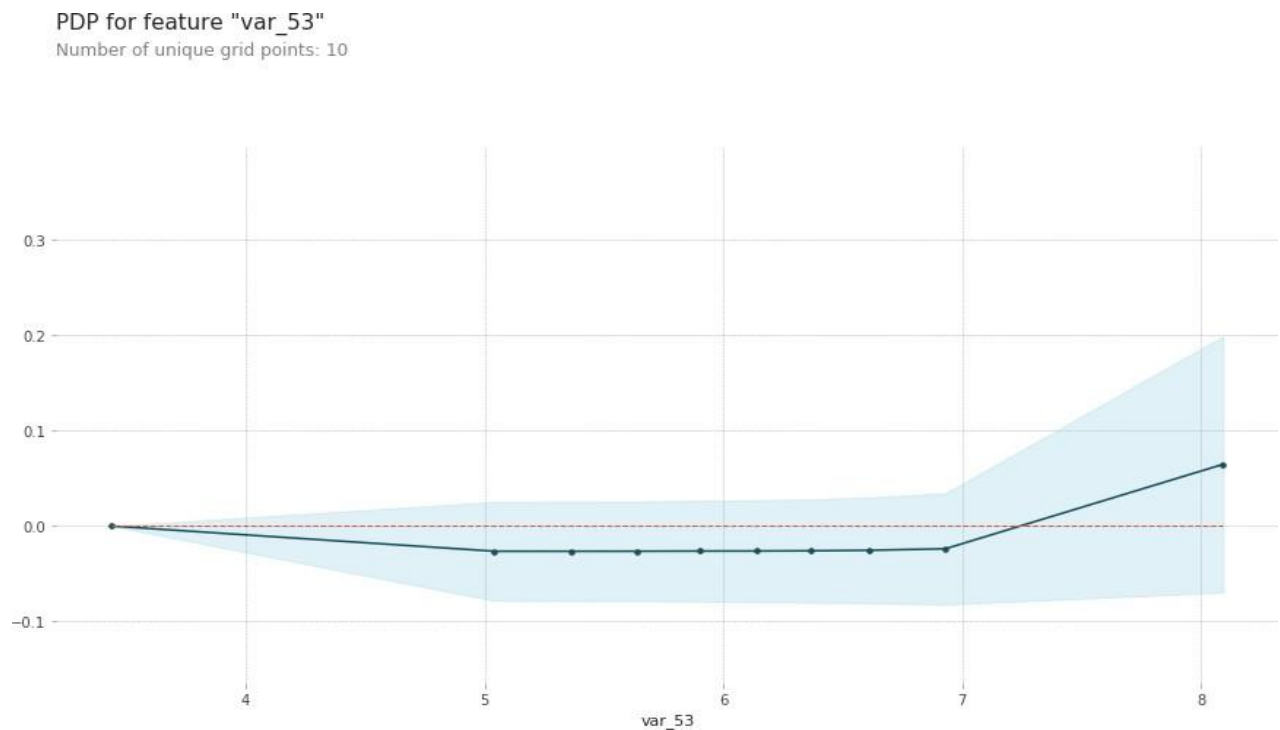
PDP for feature "var_53"
Number of unique grid points: 10

Take away:

- The y_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
- The blue shaded area indicates the level of confidence of 'var_53'.
- On y-axis having a positive value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.

```
#Create the data we will plot 'var_81'
features=[v for v in X_valid.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature=
'var_81')
#plot feature "var_81"
pdp.pdp_plot(pdp_data,'var_81')
plt.show()
```


PDP for feature "var_81"
Number of unique grid points: 10

Take away:

- The y_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
- The blue shaded area indicates the level of confidence of 'var_81'.
- On y-axis having a positive value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.

R code

```
#We will plot "var_13"
par.var_13 <- partial(rf, pred.var = c("var_13"), chull = TRUE)
plot.var_13 <- autoplot(par.var_13, contour = TRUE)
```
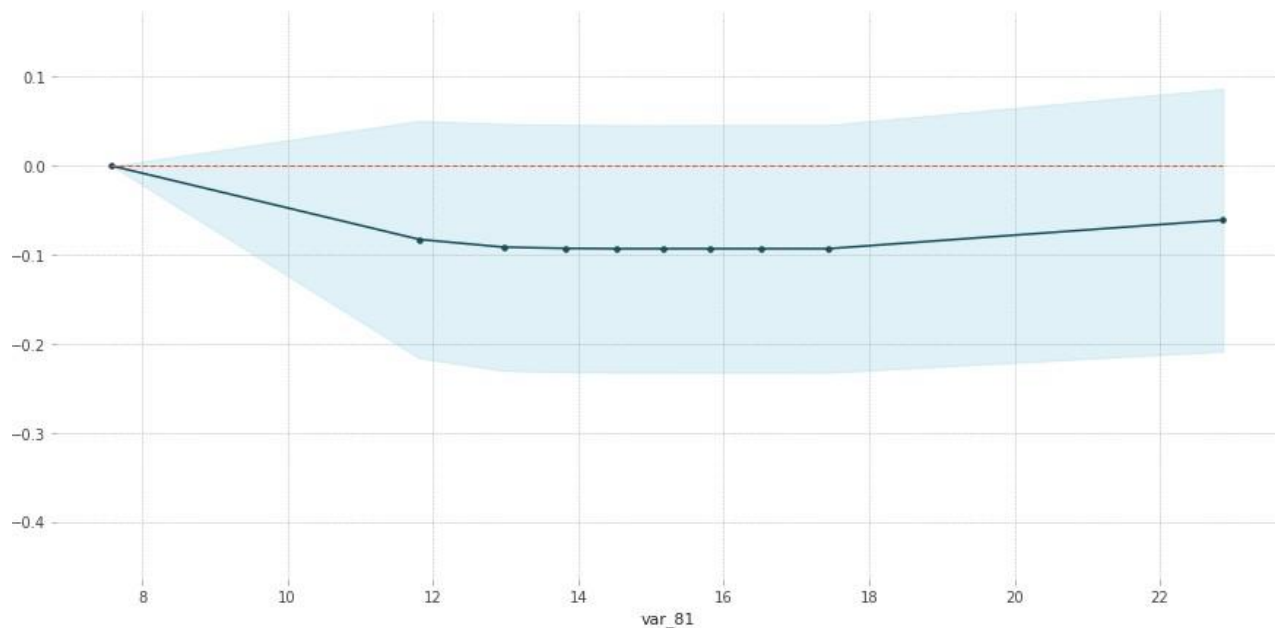

Take away:

- The y_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
- The blue shaded area indicates the level of confidence of 'var_13'.
- On y-axis having a positive value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.

```
#We will plot "var_34"
 par.var_34 <- partial(rf, pred.var = c("var_34"), chull = TRUE)
 plot.var_34 <- autoplot(par.var_34, contour = TRUE)
```
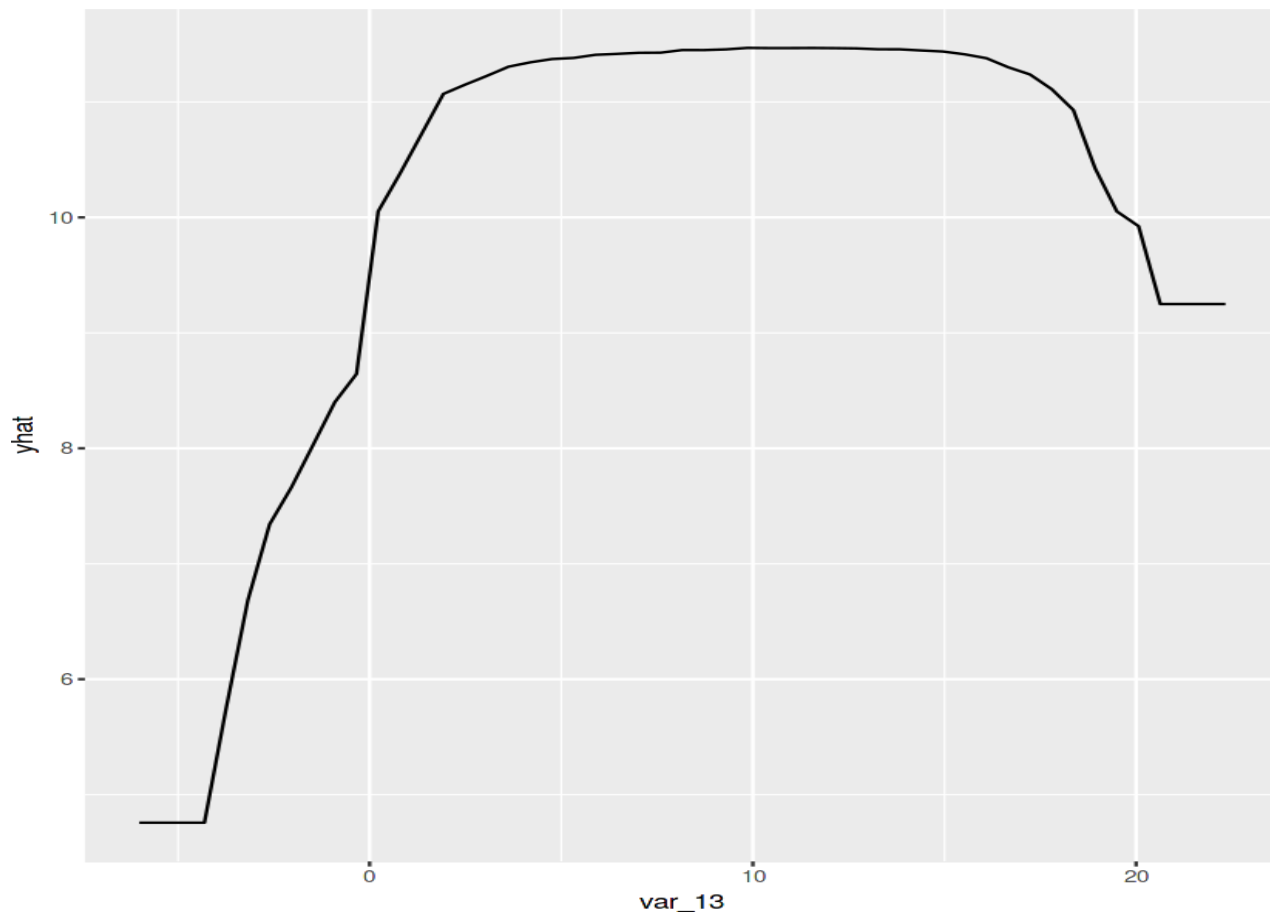

Take away:

- The y_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
- The blue shaded area indicates the level of confidence of 'var_34'.
- On y-axis having a positive value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.
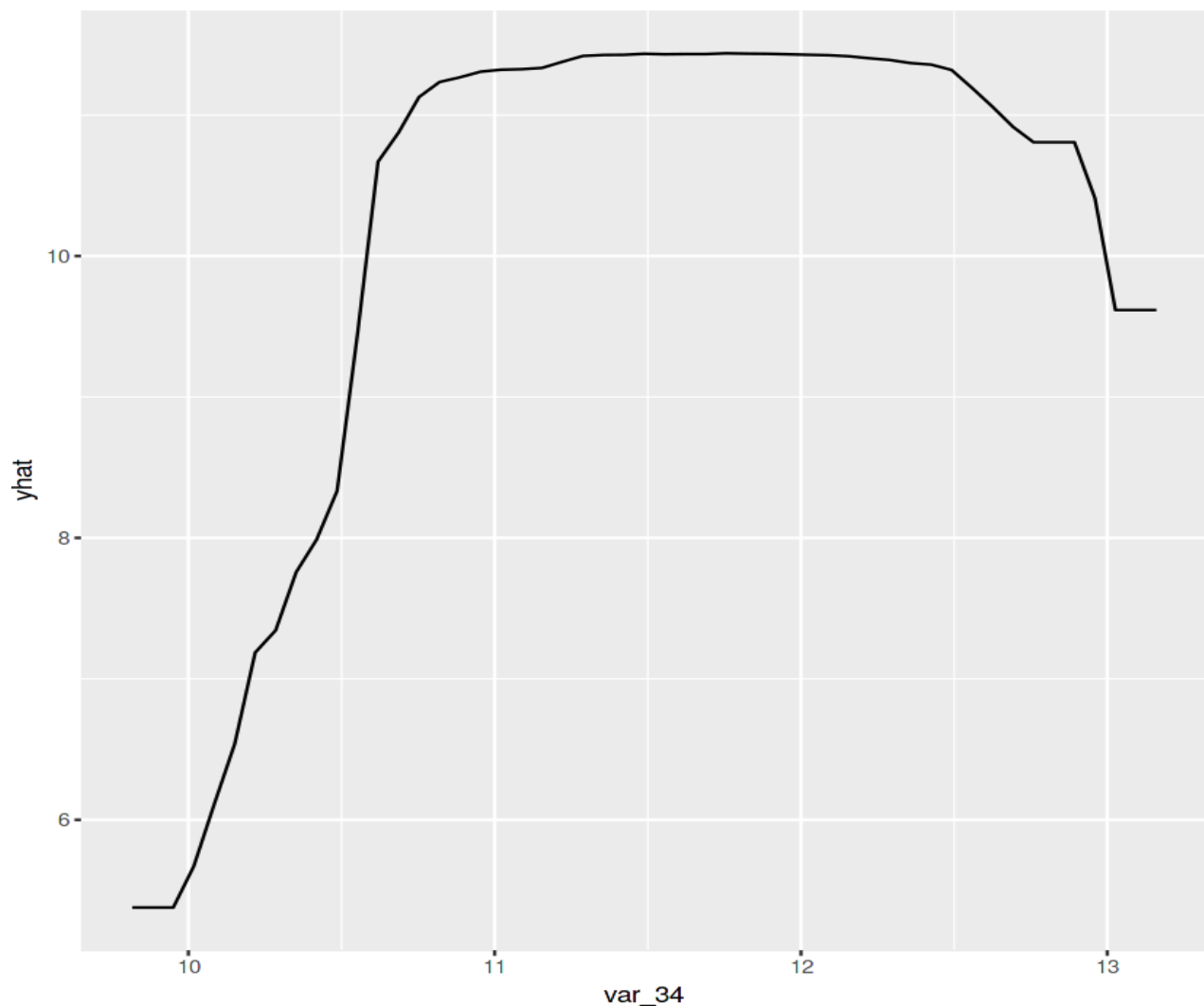
2.1 Modeling

2.1.1 Model Selection

After all early stages of preprocessing, then model the data. So, we have to select best model for this project with the help of some metrics.

The dependent variable can fall in either of the four categories:

1. Nominal
2. Ordinal
3. Interval
4. Ratio

If the dependent variable is Nominal the only predictive analysis that we can perform is **Classification**, and if the dependent variable is Interval or Ratio like this project, the normal method is to do a **Regression** analysis, or classification after binning.

Handling of imbalance data

Now we are going to explore 5 different approaches for dealing with imbalanced datasets.

- Change the performance metric
- Oversample minority class
- Under sample majority class
- Synthetic Minority Oversampling Technique(SMOTE) in Python or Random Oversampling Examples(ROSE) in R
- Change the algorithm

We always start model building from the simplest to more complex.

2.1.2 Logistic Regression

We will use a Logistic Regression to predict the values of our target variable.

Python code

```python
#Training dataset for modelling

#Training data
X=train_df.drop(['ID_code','target'],axis=1)
Y=train_df['target']

#StratifiedKFold cross validator

cv=StratifiedKFold(n_splits=5,random_state=42,shuffle=True)
for train_index,valid_index in cv.split(X,Y):
    X_train, X_valid=X.iloc[train_index], X.iloc[valid_index]
```

```
    y_train, y_valid=Y.iloc[train_index], Y.iloc[valid_index]

#Logistic regression model

lr_model=LogisticRegression(random_state=42)
#fitting the lr model
lr_model.fit(X_train,y_train)

#Accuracy of the model
lr=lr_model.score(X_train,y_train)

Accuracy of the model : 0.914

#Cross validation prediction
cv_predict=cross_val_predict(lr_model,X_valid,y_valid,cv=5)
#Cross validation score
cv_score=cross_val_score(lr_model,X_valid,y_valid,cv=5)
print('cross_val_score :',np.average(cv_score))

Cross_val_score : 0.9132

#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
lr_pred=lr_model.predict(X_test)
```

R code

Glmnet is a package that fits a generalized linear model via penalized maximum likelihood.

```
#Split the data using CreateDataPartition
train.index<-createDataPartition(train_df$target,p=0.8,list=FALSE)
train.data<-train_df[train.index,]
valid.data<-train_df[-train.index,]

#Training dataset
X_t<-as.matrix(train.data[,-c(1,2)])
y_t<-as.matrix(train.data$target)

#validation dataset
X_v<-as.matrix(valid.data[,-c(1,2)])
y_v<-as.matrix(valid.data$target)

#test dataset
test<-as.matrix(test_df[,-c(1)])

#Logistic regression model
set.seed(667)
lr_model <-glmnet(X_t,y_t, family = "binomial")
summary(lr_model)

#Cross validation prediction
```

```
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
#Plotting the missclassification error vs log(lambda) where lambda is
regularization parameter
#Minimum lambda
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)

#Model performance on validation dataset
set.seed(5363)
cv_predict.lr<-predict(cv_lr,X_v,s = "lambda.min", type = "class")

#Confusion matrix
set.seed(689)
#actual target variable
target<-valid.data$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)

#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
roc(data=valid.data[,-c(1,2)],response=target,predictor=cv_predict.lr,auc=TRUE,
plot=TRUE)
#predict the model
lr_pred<-predict(lr_model,test_df[,-c(1)],type='class')
```

Accuracy of the model is not the best metric to use when evaluating the imbalanced datasets as it may be misleading. So, we are going to change the performance metric.

Oversample minority class:

- It can be defined as adding more copies of minority class.
- It can be a good choice when we don't have a ton of data to work with.
- Drawback is that we are adding information. This may leads to overfitting and poor performance on test data.

Under sample majority class:

- It can be defined as removing some observations of the majority class.
- It can be a good choice when we have a ton of data -think millions of rows.
- Drawback is that we are removing information that may be valuable. This may leads to under fitting and poor performance on test data.

Both Oversampling and under sampling techniques have some drawbacks. So, we are not going to use this models for this problem and also we will use other best algorithms.

Synthetic Minority Oversampling Technique (SMOTE)

SMOTE uses a nearest neighbor's algorithm to generate new and synthetic data to use for training the model. In order to balance imbalanced data we are going to use SMOTE sampling method.

Python code

```python
from imblearn.over_sampling import SMOTE
#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_valid,y_valid)

#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)

smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)

Accuracy of the model : 0.798

#Cross validation prediction
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score :',np.average(cv_score))

cross_val_score : 0.80

#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
```

R code

Random Oversampling Examples (ROSE)

It creates a sample of synthetic data by enlarging the features space of minority and majority class examples. In order to balance imbalanced data we are going to use SMOTE sampling method.

```r
#Random Oversampling Examples(ROSE)
set.seed(699)
#train.data$target<-as.factor(train.data$target)
train.rose <- ROSE(target~., data =train.data[,-c(1)],seed=32)$data
table(train.rose$target)
valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=42)$data
table(valid.rose$target)
```

```r
#Baseline logistic regression model
set.seed(462)
lr_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target),
family = "binomial")
summary(lr_rose)

#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),
family = "binomial", type.measure = "class")

#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)

#Model performance on validation dataset
set.seed(442)
cv_predict.rose<-predict(cv_rose,as.matrix(valid.rose),s = "lambda.min",
type = "class")
cv_predict.rose

#Confusion matrix
set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.rose<-as.factor(cv_predict.rose)
confusionMatrix(data=cv_predict.rose,reference=target)

#ROC_AUC score and curve
set.seed(843)
cv_predict.rose<-as.numeric(cv_predict.rose)
roc(data=valid.rose[,-c(1,2)],response=target,predictor=cv_predict.rose,auc=TRUE,
plot=TRUE)

#predict the model
set.seed(6543)
rose_pred<-predict(lr_rose,test_df[,-c(1)],type='class')
```

LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. We are going to use LightGBM model.

Python code

Let us build LightGBM model

```python
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)

#Selecting best hyper parameters by tuning of different parameters

params={'boosting_type': 'gbdt',
        'max_depth' : -1, #no limit for max_depth if <0
        'objective': 'binary',
        'boost_from_average':False,
        'nthread': 8,
        'metric':'auc',
        'num_leaves': 100,
        'learning_rate': 0.03,
        'max_bin': 950,        #default 255
        'subsample_for_bin': 200,
        'subsample': 1,
        'subsample_freq': 1,
        'colsample_bytree': 0.8,
        'reg_alpha': 1.2, #L1 regularization(>0)
        'reg_lambda': 1.2,#L2 regularization(>0)
        'min_split_gain': 0.5, #>0
        'min_child_weight': 1,
        'min_child_samples': 5,
        'is_unbalance':True,
        }

 num_rounds=3000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],
verbose_eval=100,early_stopping_rounds = 1000)

X_test=test_df.drop(['ID_code'],axis=1)
#predict the model
#probability predictions
lgbm_predict_prob=lgbm.predict(X_test,random_state=42,
num_iteration=lgbm.best_iteration)
#Convert to binary output 1 or 0
lgbm_predict=np.where(lgbm_predict_prob>=0.5,1,0)
```

```
Training until validation scores don't improve for 5000 rounds.
[1000]  training's auc: 0.939079         valid_1's auc: 0.882655
[2000]  training's auc: 0.958502         valid_1's auc: 0.887842
[3000]  training's auc: 0.971937         valid_1's auc: 0.889724
[4000]  training's auc: 0.981492         valid_1's auc: 0.890474
[5000]  training's auc: 0.988242         valid_1's auc: 0.890772
[6000]  training's auc: 0.992813         valid_1's auc: 0.890549
[7000]  training's auc: 0.995775         valid_1's auc: 0.890488
[8000]  training's auc: 0.997627         valid_1's auc: 0.890549
[9000]  training's auc: 0.998739         valid_1's auc: 0.890309
[10000] training's auc: 0.999359         valid_1's auc: 0.889882
Did not meet early stopping. Best iteration is:
[10000] training's auc: 0.999359         valid_1's auc: 0.889882


<lightgbm.basic.Booster at 0x7f8b795edac8>
```

R code

```
#Convert data frame to matrix
X_train<-as.matrix(train_data[,-c(1,2)])
y_train<-as.matrix(train_data$target)
X_valid<-as.matrix(valid_data[,-c(1,2)])
y_valid<-as.matrix(valid_data$target)
test_data<-as.matrix(test_df[,-c(1)])

#training data
lgb.train <- lgb.Dataset(data=X_train, label=y_train)
#Validation data
lgb.valid <- lgb.Dataset(data=X_valid,label=y_valid)



#Choosing parameters

lgb.grid = list(objective = "binary",
                metric = "auc",
                boost ="gbdt"
                min_sum_hessian_in_leaf = 1,
                feature_fraction = 0.7,
                bagging_fraction = 0.7,
                bagging_freq = 5,
                learning_rate=0.05,
                num_leaves=80,
                num_threads=10,
                min_data = 100,
                max_bin = 200,
                lambda_l1 = 8,
```

```
               lambda_l2 = 1.3,
               min_data_in_bin=150,
               min_gain_to_split = 20,
               min_data_in_leaf = 40,
               is_unbalance = TRUE)

 lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,
eval_freq =1000,valids=list(val1=lgb.train,val2=lgb.valid),early_stopping_rounds =
5000)
```

```
[1]:    val1's auc:0.594625     val2's auc:0.586947
[1001]: val1's auc:0.922163     val2's auc:0.895506
[2001]: val1's auc:0.928038     val2's auc:0.898477
[3001]: val1's auc:0.929811     val2's auc:0.899515
[4001]: val1's auc:0.930609     val2's auc:0.899988
[5001]: val1's auc:0.931106     val2's auc:0.900336
[6001]: val1's auc:0.931368     val2's auc:0.900598
[7001]: val1's auc:0.931604     val2's auc:0.900685
[8001]: val1's auc:0.931813     val2's auc:0.900826
[9001]: val1's auc:0.931982     val2's auc:0.900869
[10000]:        val1's auc:0.93215      val2's auc:0.900892
```

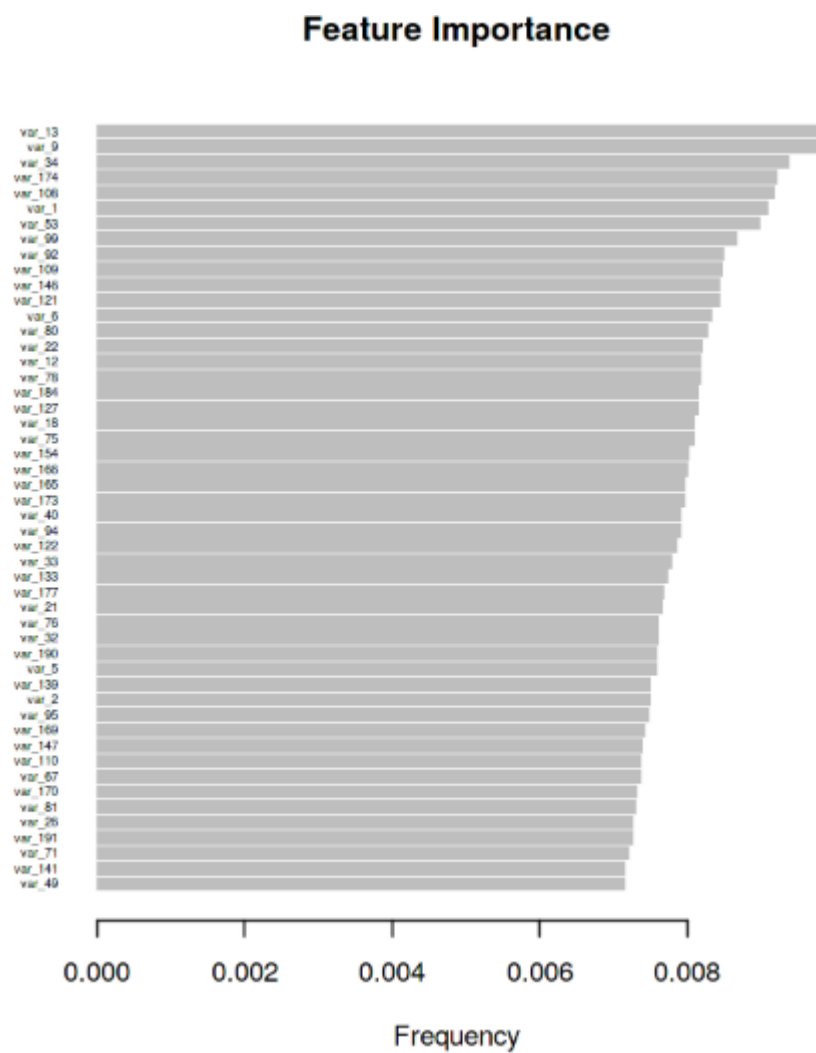## Important features plot

Python code

```
 #plot the important features
lgb.plot_importance(lgbm,max_num_features=150,importance_type="split",figsize=(20,
50))
```


26
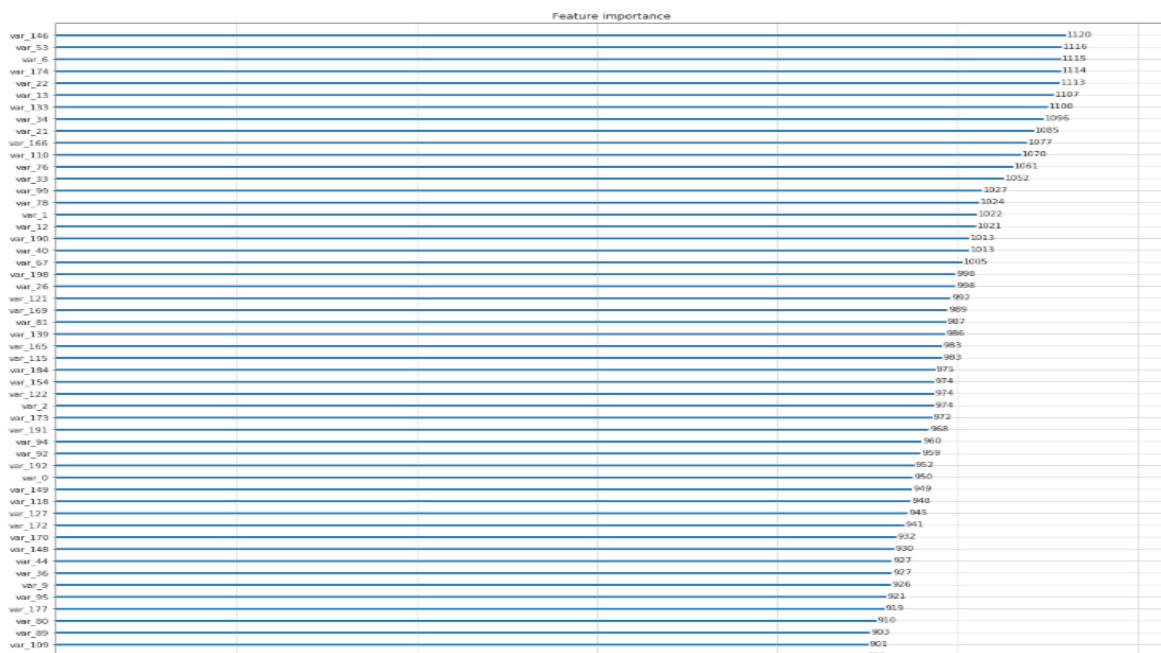
R code

```
tree_imp <- lgb.importance(lgbm.model, percentage = TRUE)
lgb.plot.importance(tree_imp, top_n = 50, measure = "Gain")
```

Feature Importance

Chapter 3

Conclusion

3.1   Model Evaluation

Now, we have a three models for predicting the target variable, but we need to decide which model better for this project. There are many metrics used for model evaluation. Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

For classification problems, the confusion matrix used for evaluation. But, in our case the data is imbalanced. So, roc_auc_score is used for evaluation.

In this project, we are using two metrics for model evaluation as follows,

Confusion Matrix: - It is a technique for summarizing the performance of a classification algorithm.

The number of correct predictions and incorrect predictions are summarized with count values and

broken down by each class.

Accuracy: - The ratio of correct predictions to total predictions

$$\text{Accuracy} = \frac{TP+TN}{Total\ Predictions}$$

Misclassification error: - The ratio of incorrect predictions to total predictions

$$\text{Error rate} = \frac{FN+FP}{Total\ predictions}$$

Accuracy=1-Error rate

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP+FN} \leftrightarrow \text{Recall}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{True Negative Rate (TNR)} = \frac{TN}{TN+FP} \leftrightarrow \text{Specificity}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP+TN}$$

$$\text{False Negative rate (FNR)} = \frac{FN}{FN+TP}$$

F1 score :- Harmonic mean of precision and recall, used to indicate balance between them.

$$\text{F1 score} = \frac{2*Precision*Recall}{Precision+Recall}$$
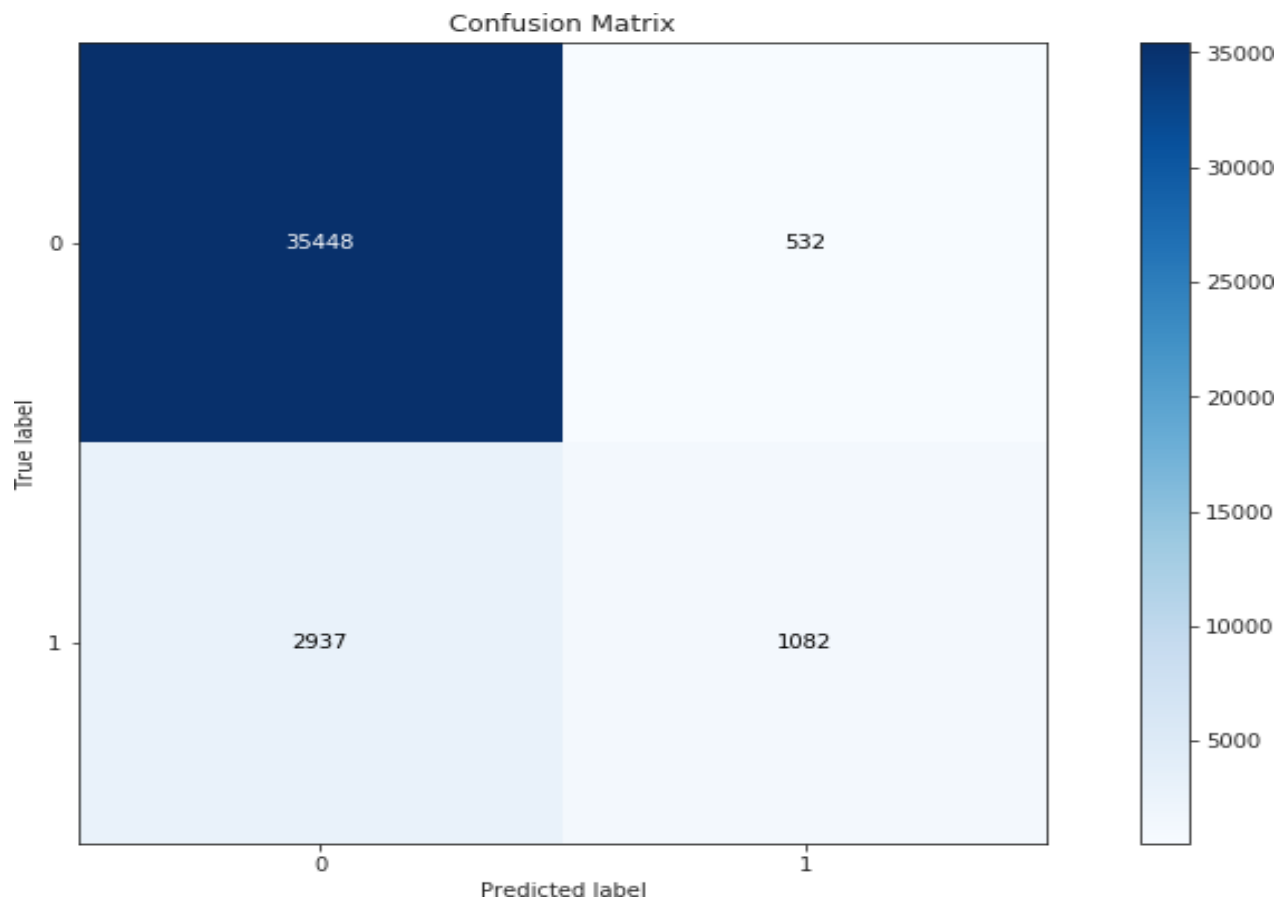
Receiver operating characteristics (ROC)_Area under curve(AUC) Score

roc_auc_score :- It is a metric that computes the area under the Roc curve and also used metric for imbalanced data.

Roc curve is plotted true positive rate or Recall on y axis against false positive rate or specificity on x axis. The larger the area under the roc curve better the performance of the model.

Logistic Regression

```
#Confusion matrix
cm=confusion_matrix(y_valid,cv_predict)
#Plot the confusion matrix
plot_confusion_matrix(y_valid,cv_predict,normalize=False,figsize=(15,8))
```


Confusion Matrix

```
#ROC_AUC curve

plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_valid,cv_predict)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```
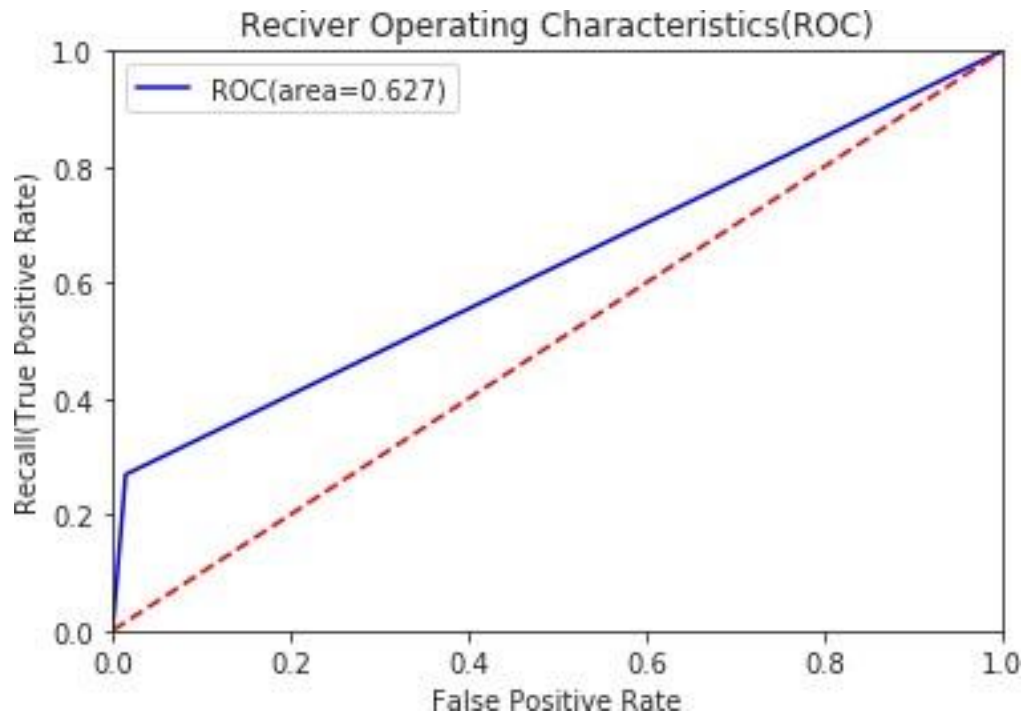
When we compare the roc_auc_score and cross validation score, conclude that model is not performing well on imbalanced data.

Classification report

```
#Classification report
scores=classification_report(y_valid,cv_predict)
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     35980
           1       0.67      0.27      0.38      4019

   micro avg       0.91      0.91      0.91     39999
   macro avg       0.80      0.63      0.67     39999
weighted avg       0.90      0.91      0.90     39999
```

We can observed that f1 score is high for number of customers those who will not make a transaction then who will make a transaction. So, we are going to change the algorithm.

R code

Logistic Regression

```
#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")

#Plotting misclassification error vs log(lambda)
#Minimum lambda-Regularization parameter
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)
```
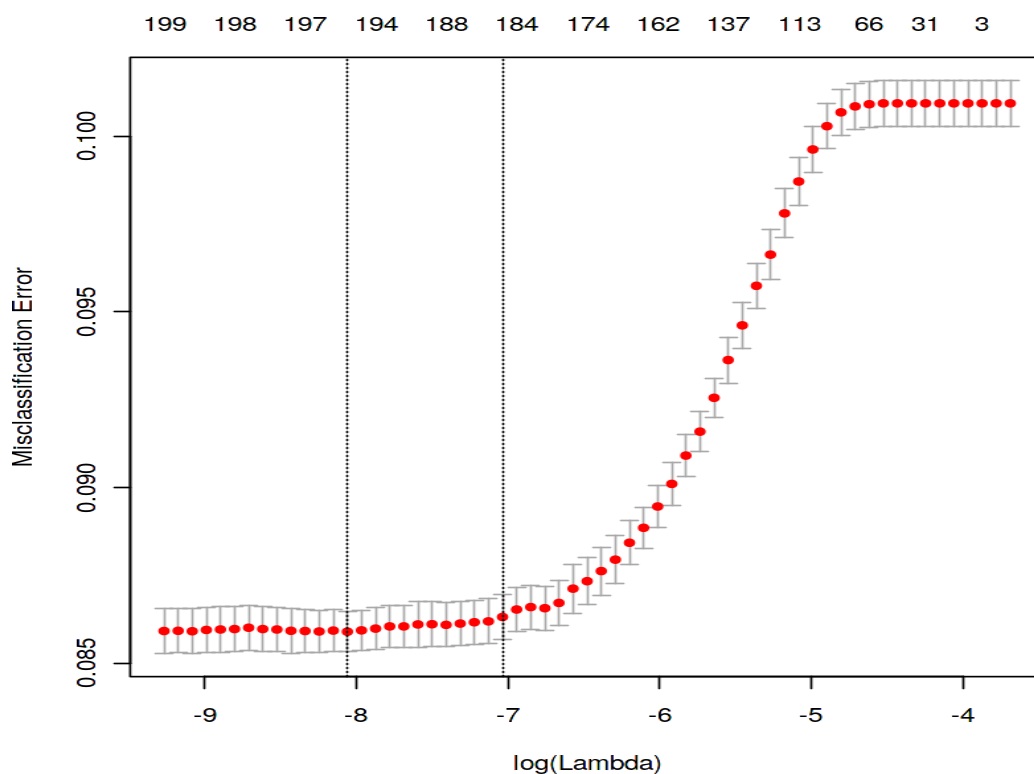

We can observed that miss classification error increases as increasing the log(Lambda).

```
#Confusion matrix
set.seed(689)
#actual target variable
target<-valid.data$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     1     2
         1 35618  2973
         2   434   975

               Accuracy : 0.9148
                 95% CI : (0.912, 0.9175)
    No Information Rate : 0.9013
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3292

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9880
            Specificity : 0.2470
         Pos Pred Value : 0.9230
         Neg Pred Value : 0.6920
             Prevalence : 0.9013
         Detection Rate : 0.8904
   Detection Prevalence : 0.9648
      Balanced Accuracy : 0.6175

       'Positive' Class : 1
```

Reciever operating characteristics(ROC)-Area under curve(AUC) score and curve

```
#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
roc(data=valid.data[,-c(1,2)],response=target,predictor=cv_predict.lr,auc=TRUE,
plot=TRUE)
```
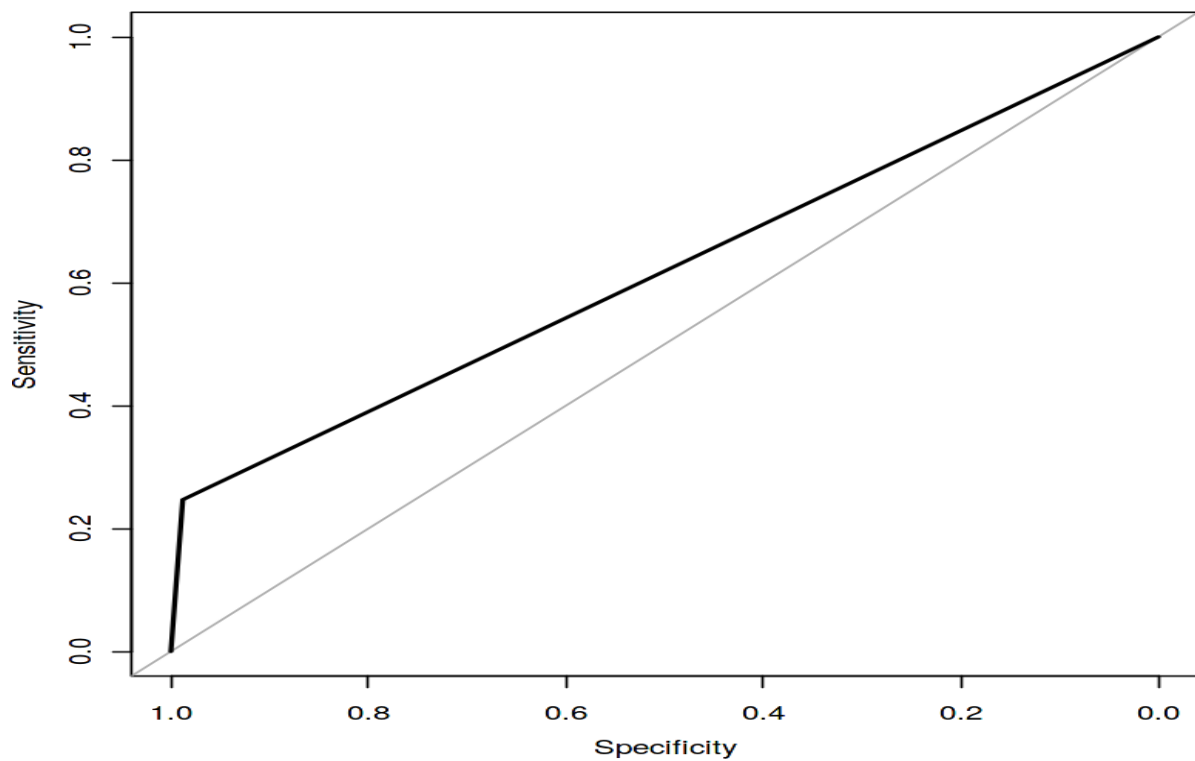
```
Call:
roc.default(response = target, predictor = cv_predict.lr, auc = TRUE,      plot = TRUE, data = v
alid.data[, -c(1, 2)])

Data: cv_predict.lr in 36052 controls (target 1) < 3948 cases (target 2).
Area under the curve: 0.6175
```
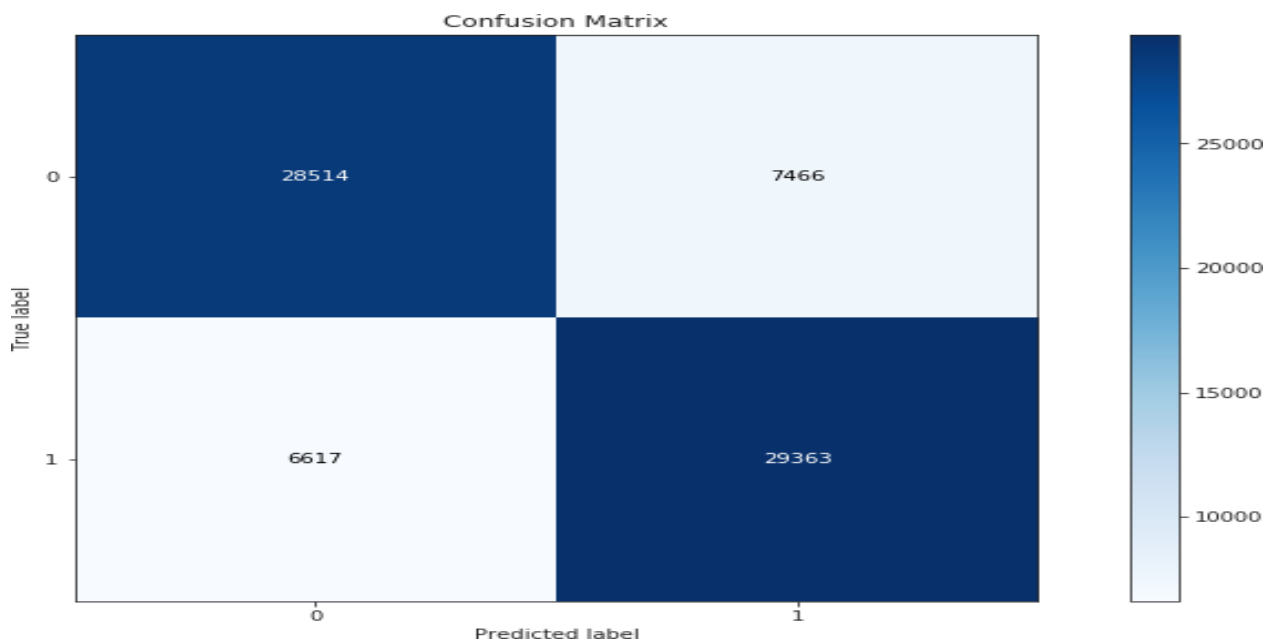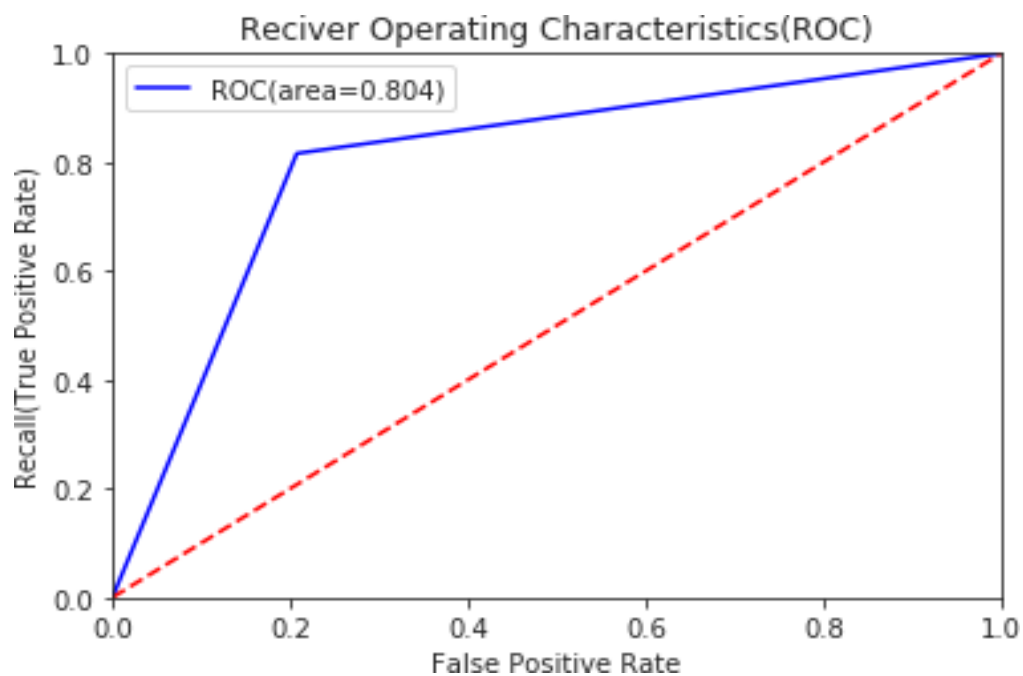
Python code

Synthetic Minority Oversampling Technique (SMOTE)

```
#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
#Plot the confusion matrix
plot_confusion_matrix(y_smote_v,cv_pred,normalize=False,figsize=(15,8))
```

Reciever operating characteristics (ROC)-Area under curve (AUC) score and curve

```
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_smote_v,cv_pred)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

Classification report

```
#Classification report
scores=classification_report(y_smote_v,cv_pred)
print(scores)
```

```
             precision    recall  f1-score   support

          0       0.81      0.79      0.80     35980
          1       0.80      0.82      0.81     35980

  micro avg       0.80      0.80      0.80     71960
  macro avg       0.80      0.80      0.80     71960
weighted avg       0.80      0.80      0.80     71960
```

We can observed that smote model is performing well on imbalance data compare to baseline logistic regression.

R code
Random Oversampling Examples (ROSE)

```
#Plotting misclassification error vs log(lambda)
#lambda-Regularization parameter
#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)
```
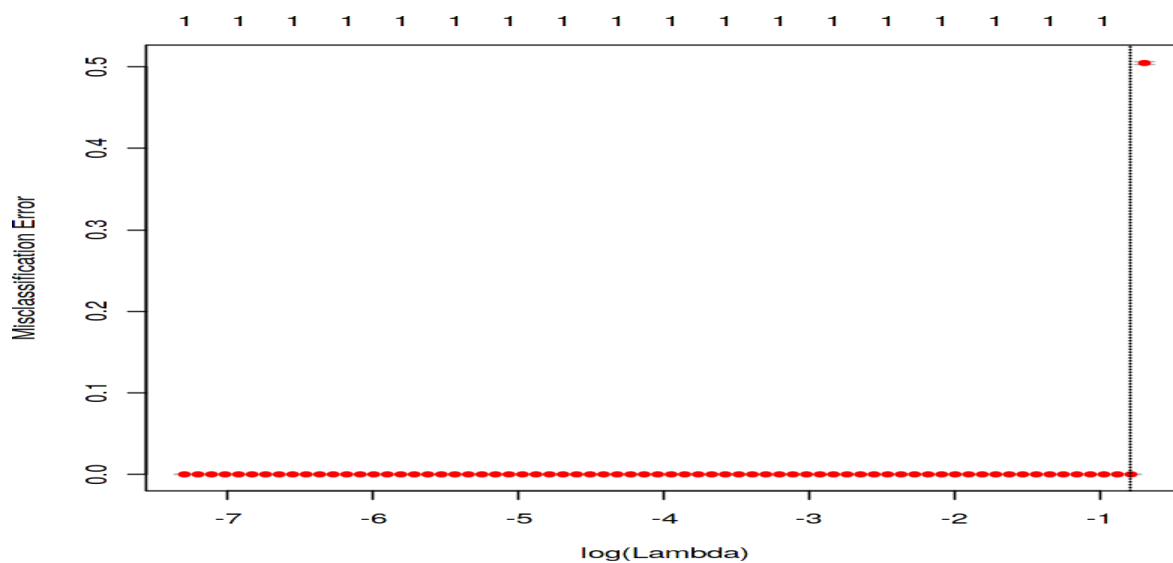
```
#Confusion matrix
set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.rose<-as.factor(cv_predict.rose)
#Confusion matrix
confusionMatrix(data=cv_predict.rose,reference=target)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     1     2
         1 20012     0
         2     0 19988

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.5003
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5003
         Detection Rate : 0.5003
   Detection Prevalence : 0.5003
      Balanced Accuracy : 1.0000

       'Positive' Class : 1
```

Reciever operating characteristics (ROC)-Area under curve(AUC) score and curve

```
#ROC_AUC score and curve
set.seed(843)
#convert to numeric
cv_predict.rose<-as.numeric(cv_predict.rose)
```
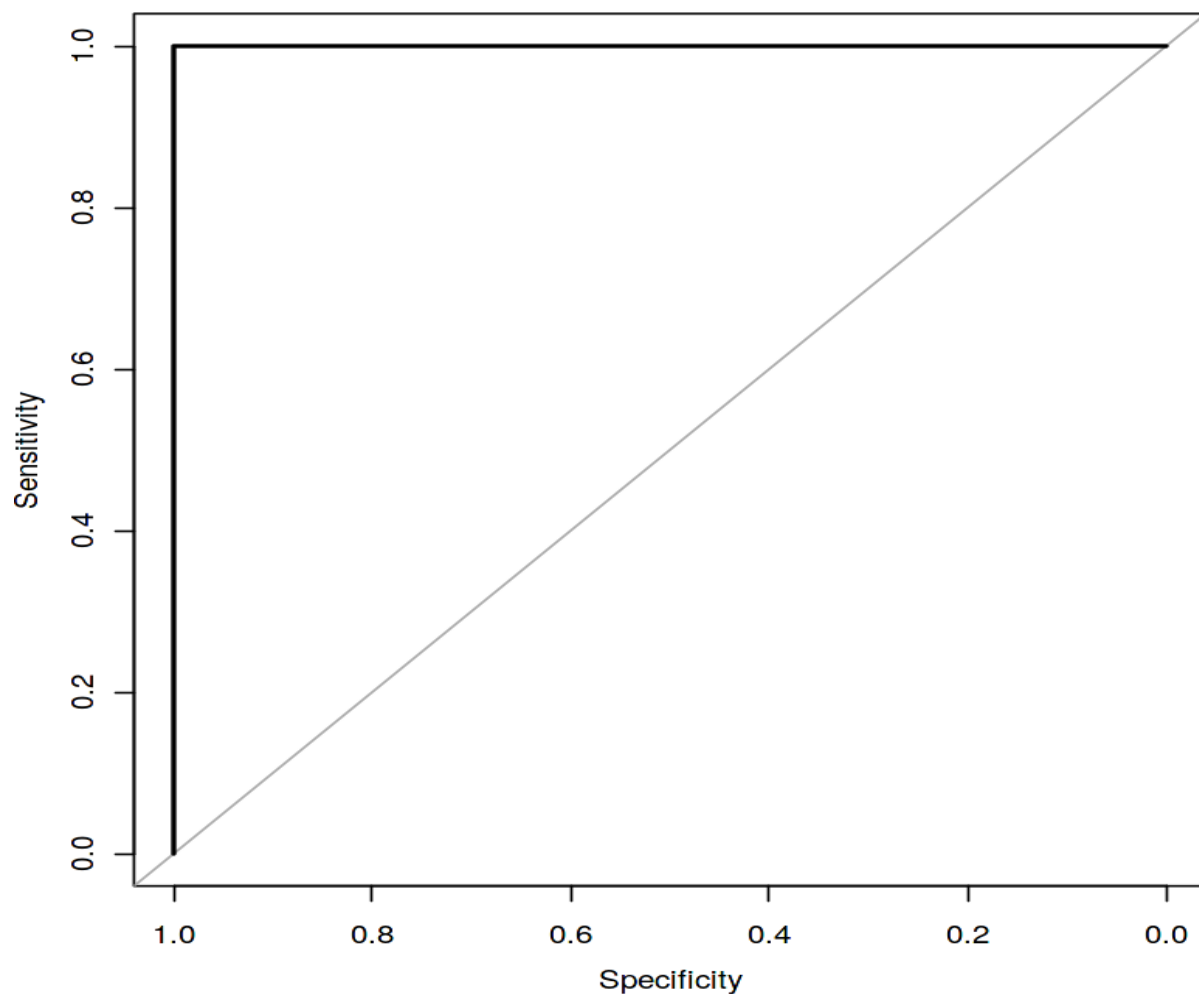
```
roc(data=valid.rose[,-c(1,2)],response=target,predictor=cv_predict.rose,auc=TR
UE,
 plot=TRUE)
```

```
Call:
roc.default(response = target, predictor = cv_predict.rose, auc = TRUE,       plot = TRUE, data =
valid.rose[, -c(1, 2)])

Data: cv_predict.rose in 20012 controls (target 1) < 19988 cases (target 2).
Area under the curve: 1
```


I tried different ways to get good accuracy like changing count of one target class variable.
Finally got area under ROC curve is 1 but this may not be possible.

3.2   Model Selection

When we compare scores of area under the ROC curve of all the models for an imbalanced data. We could conclude that below points as follow,

1. Logistic regression model is not performed well on imbalanced data.
2. We balance the imbalanced data using resampling techniques like SMOTE in python and ROSE in R.
3. Baseline logistic regression model is performed well on balanced data.
4. LightGBM model performed well on imbalanced data.

Finally LightGBM is best choice for identifying which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

Appendix A - Extra Figures

ggploт2 vısualizations

Target classes count

Distribution of train attributes

var_1

var_2
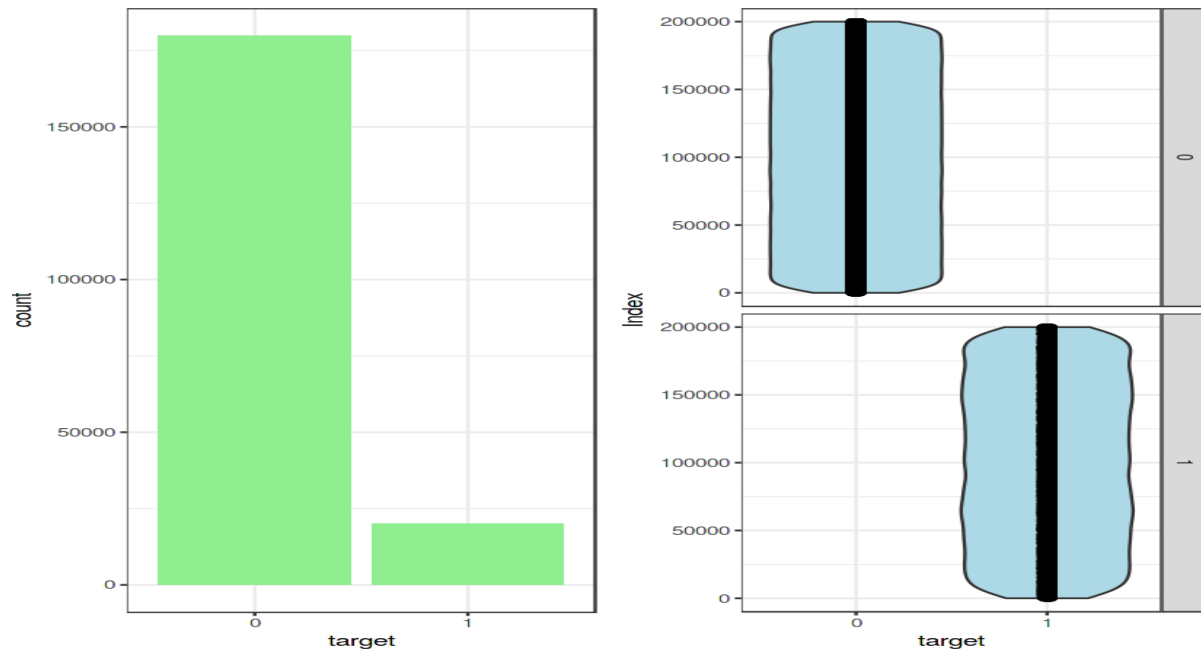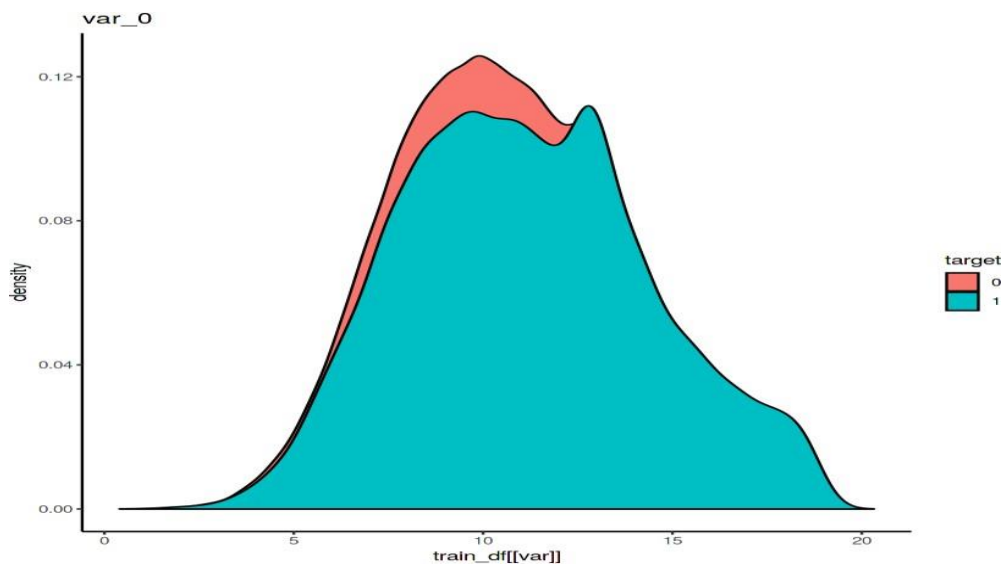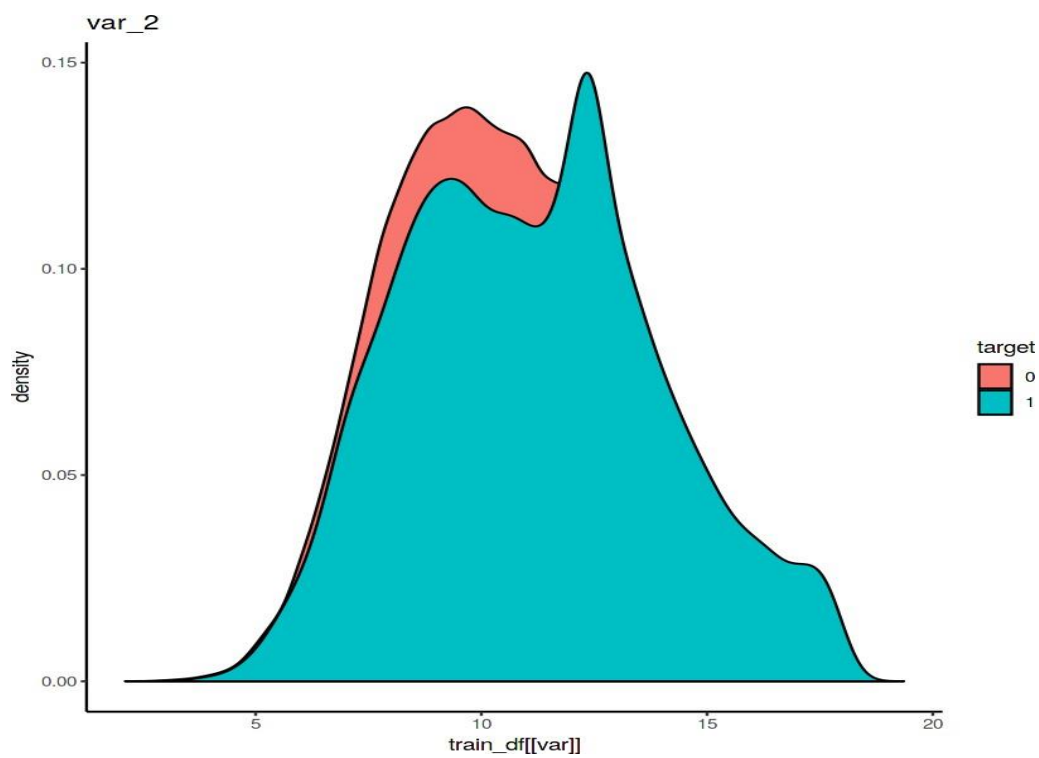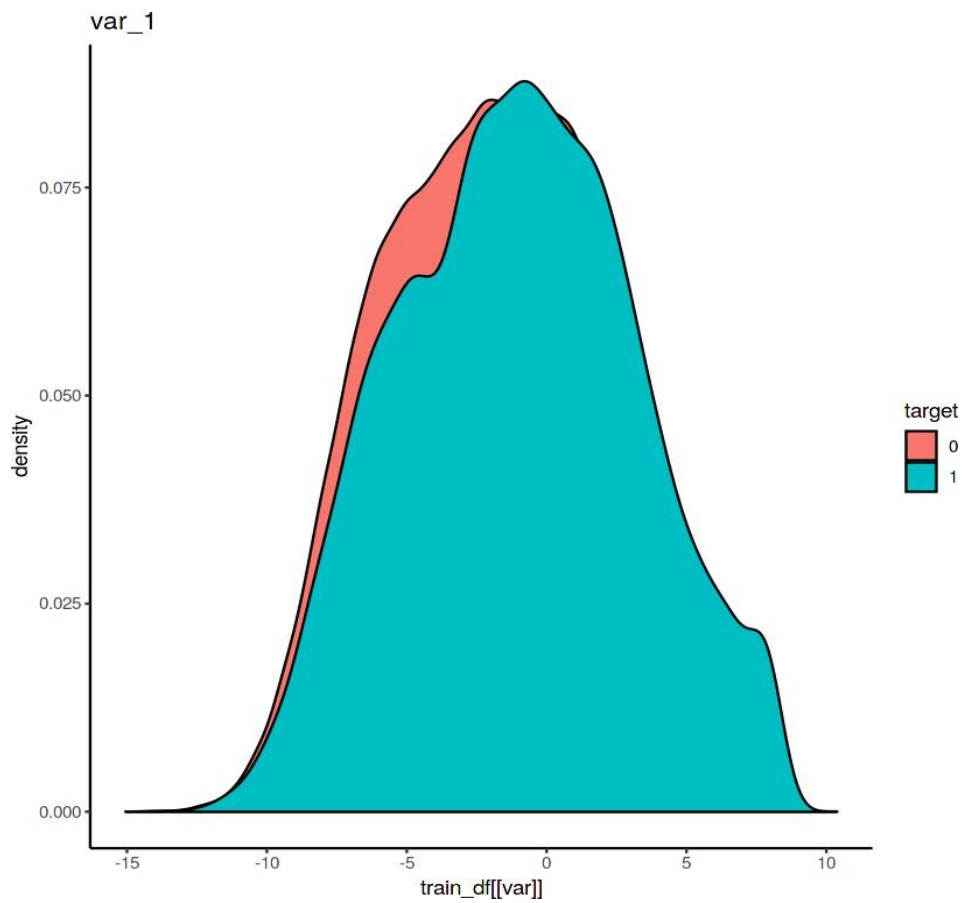
Distribution of test attribiutes
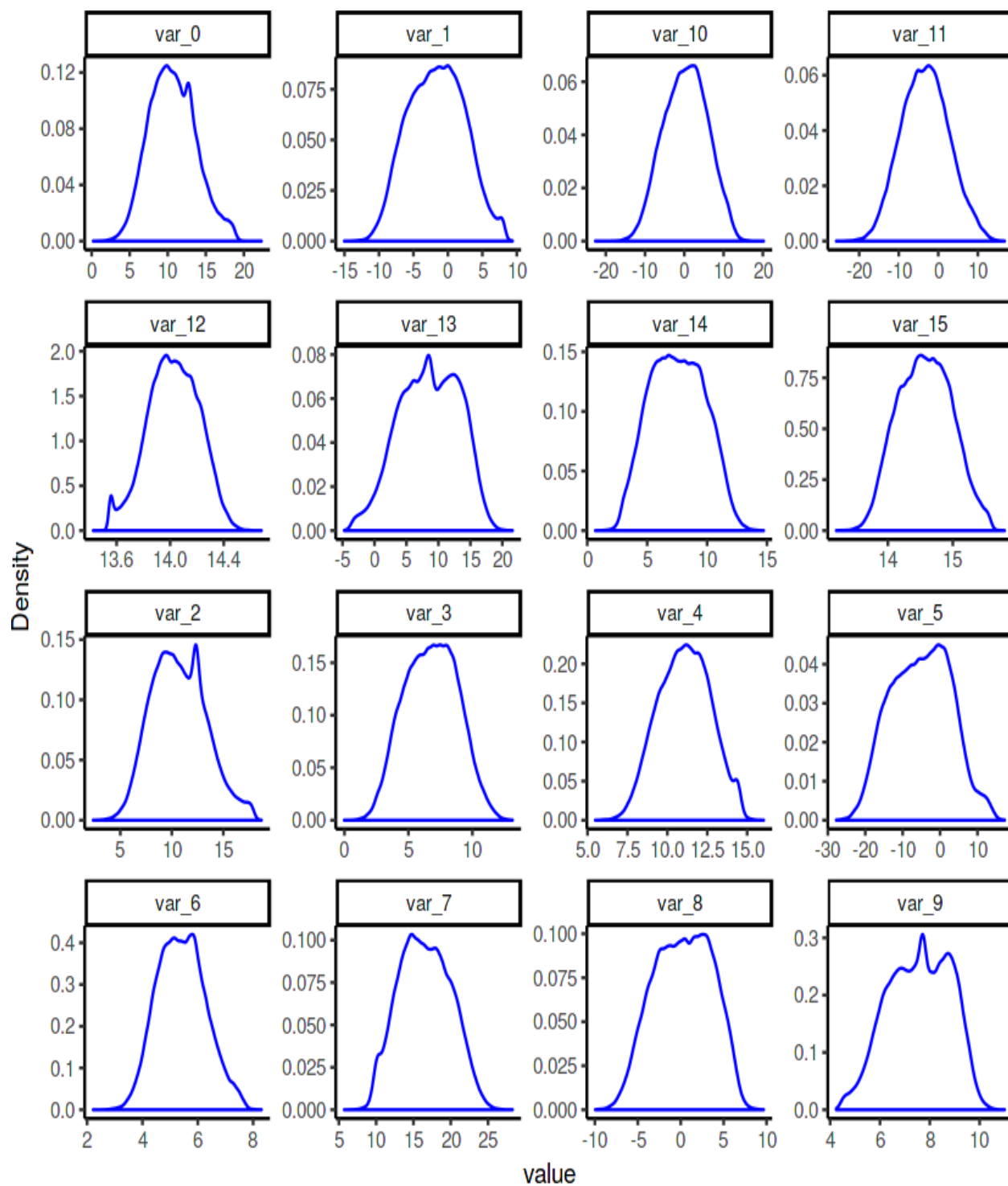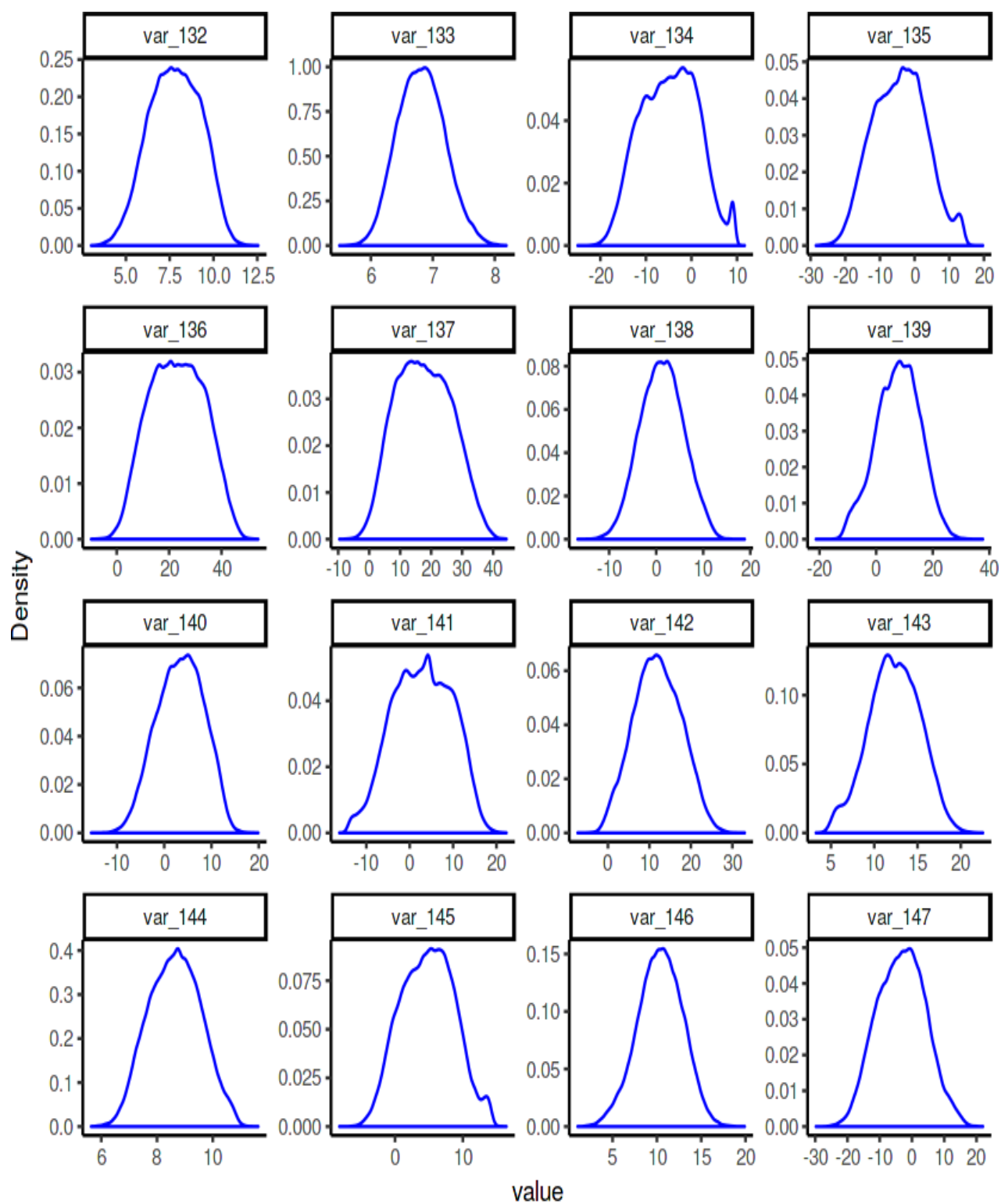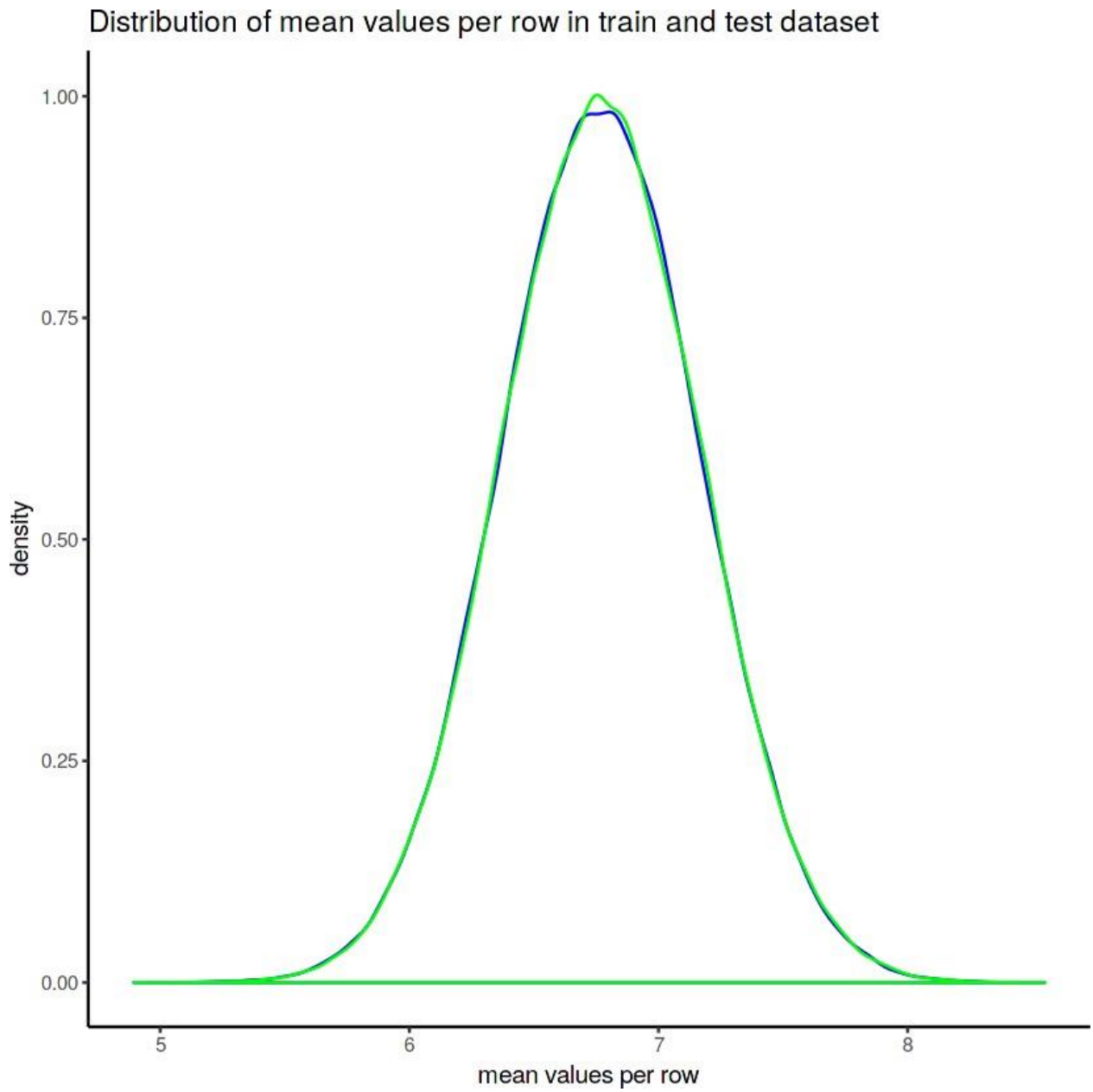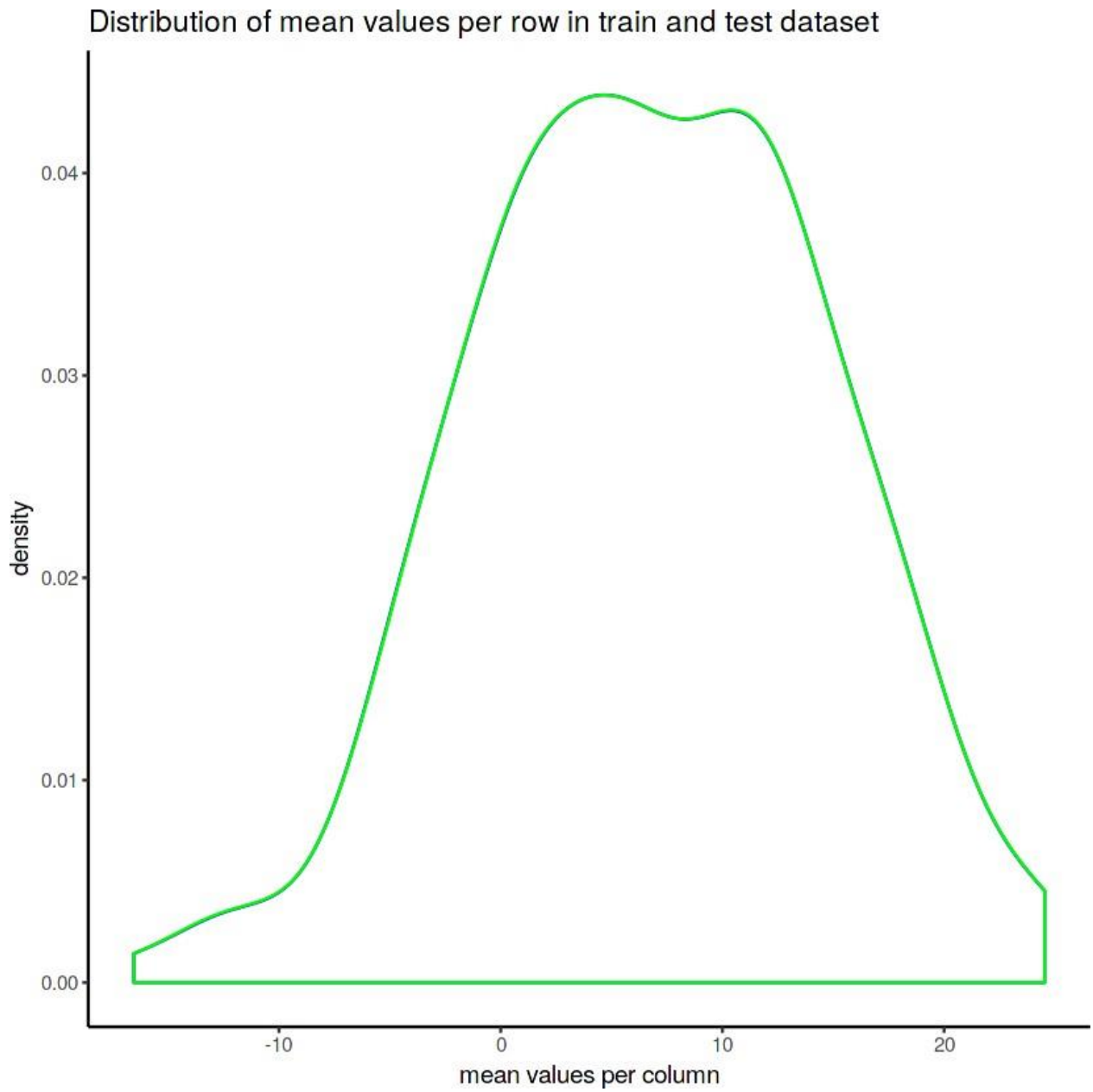
Distribution of mean values per row in train and test dataset
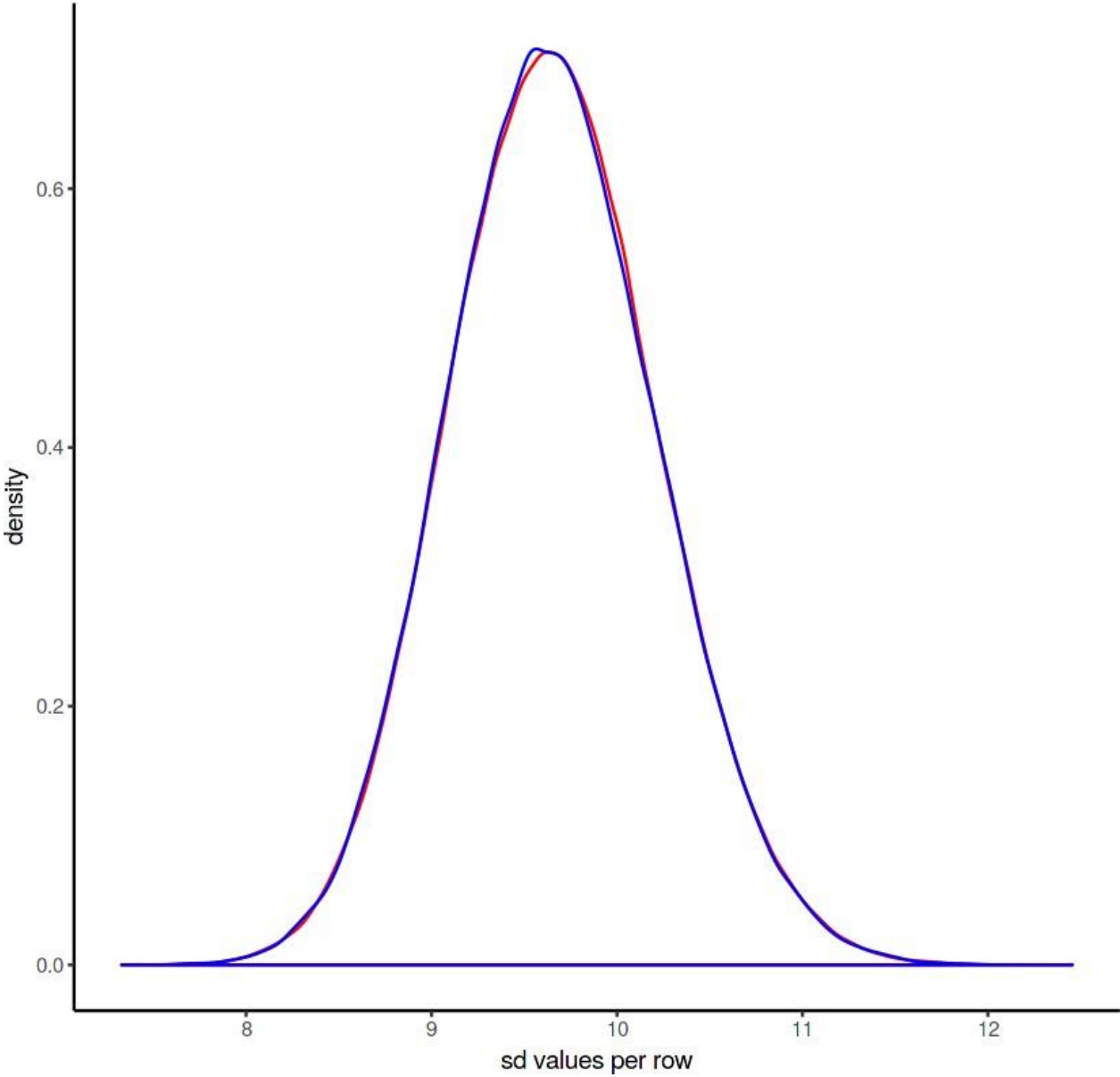
Distribution of mean values per row in train and test dataset

Distribution of sd values per row in train and test dataset

Distribution of std values per column in train and test dataset

Distribution of skewness values per row in train and test dataset
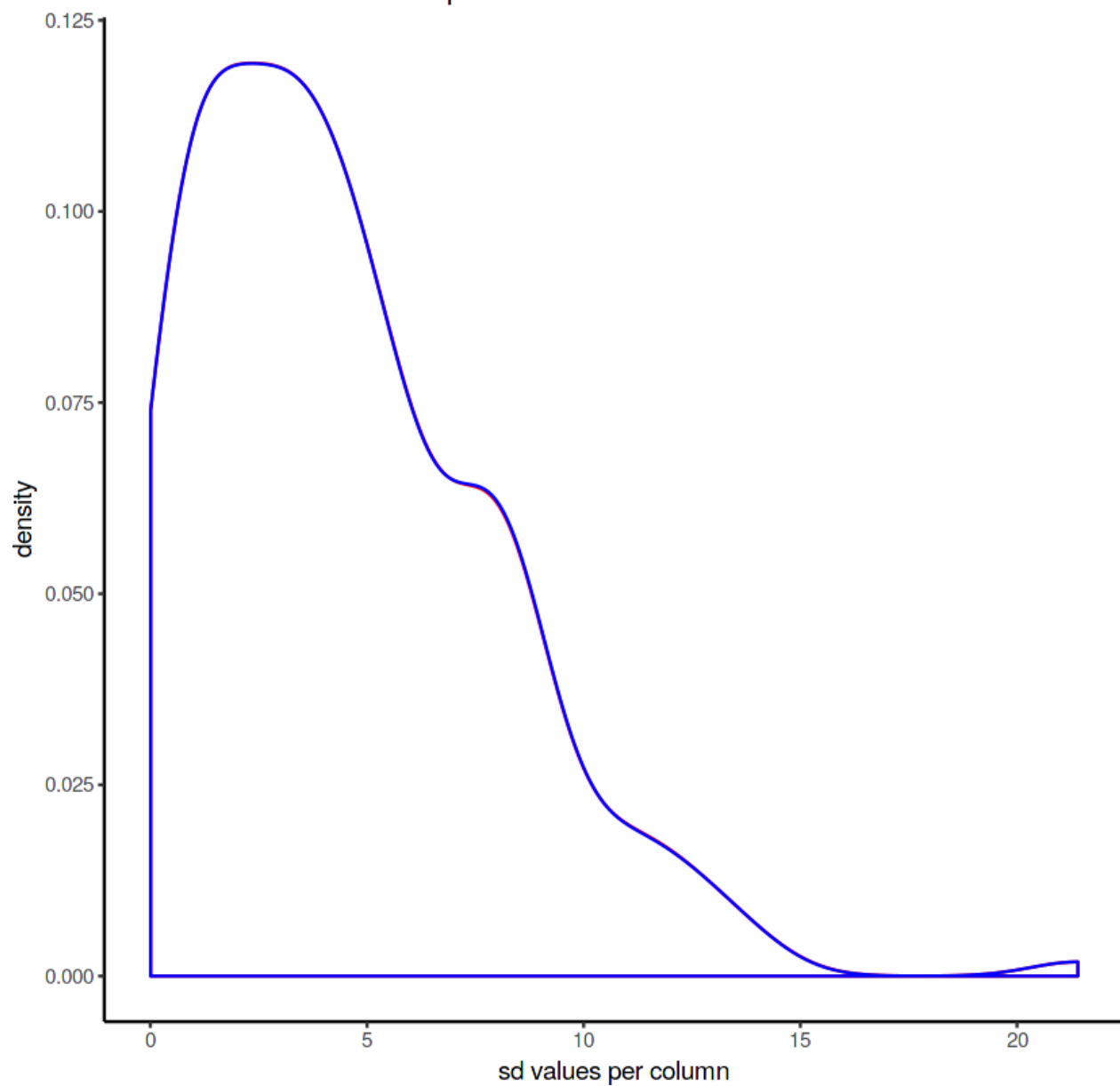
Distribution of skewness values per column in train and test dataset
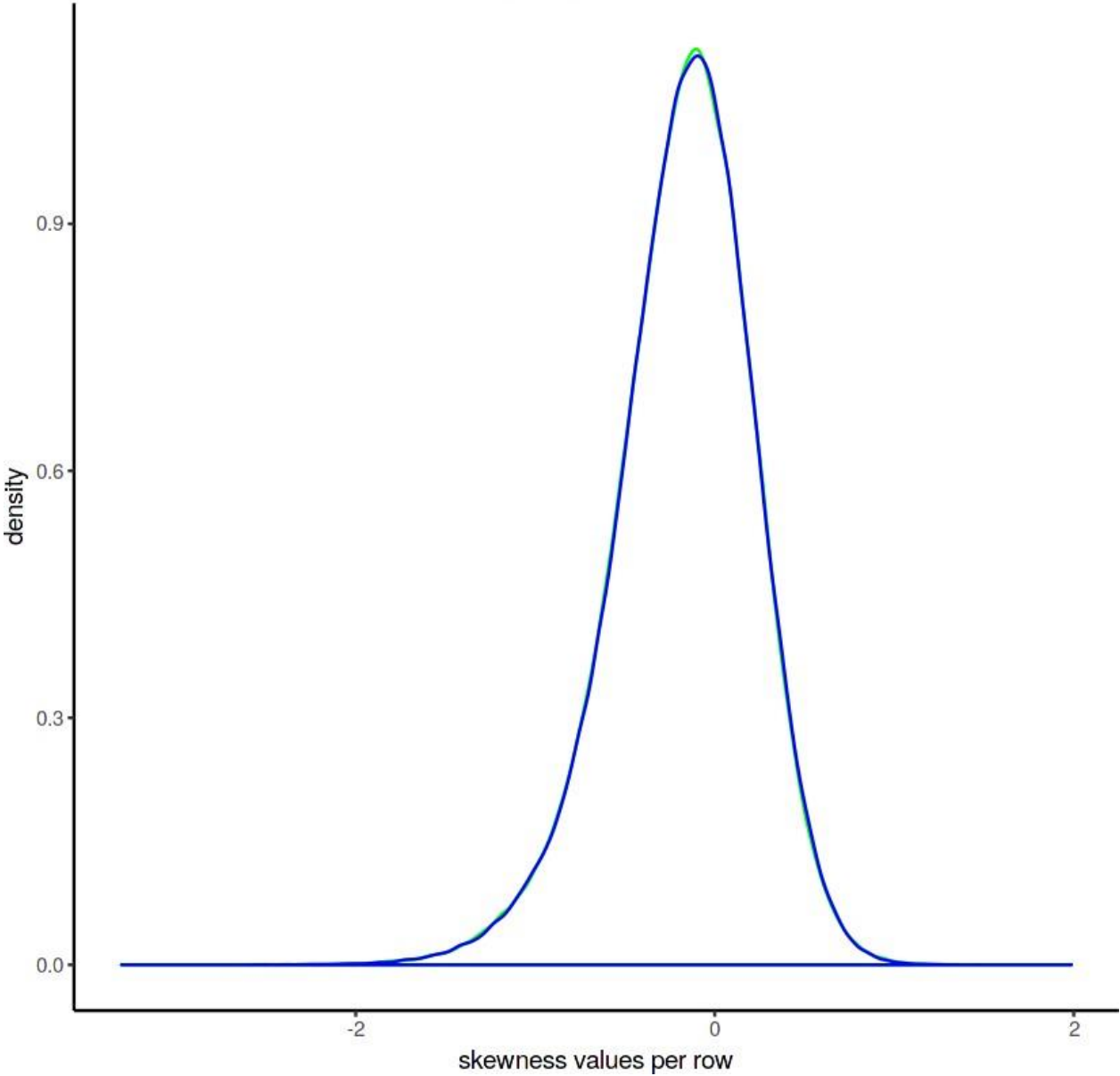
Distribution of kurtosis values per row in train and test dataset
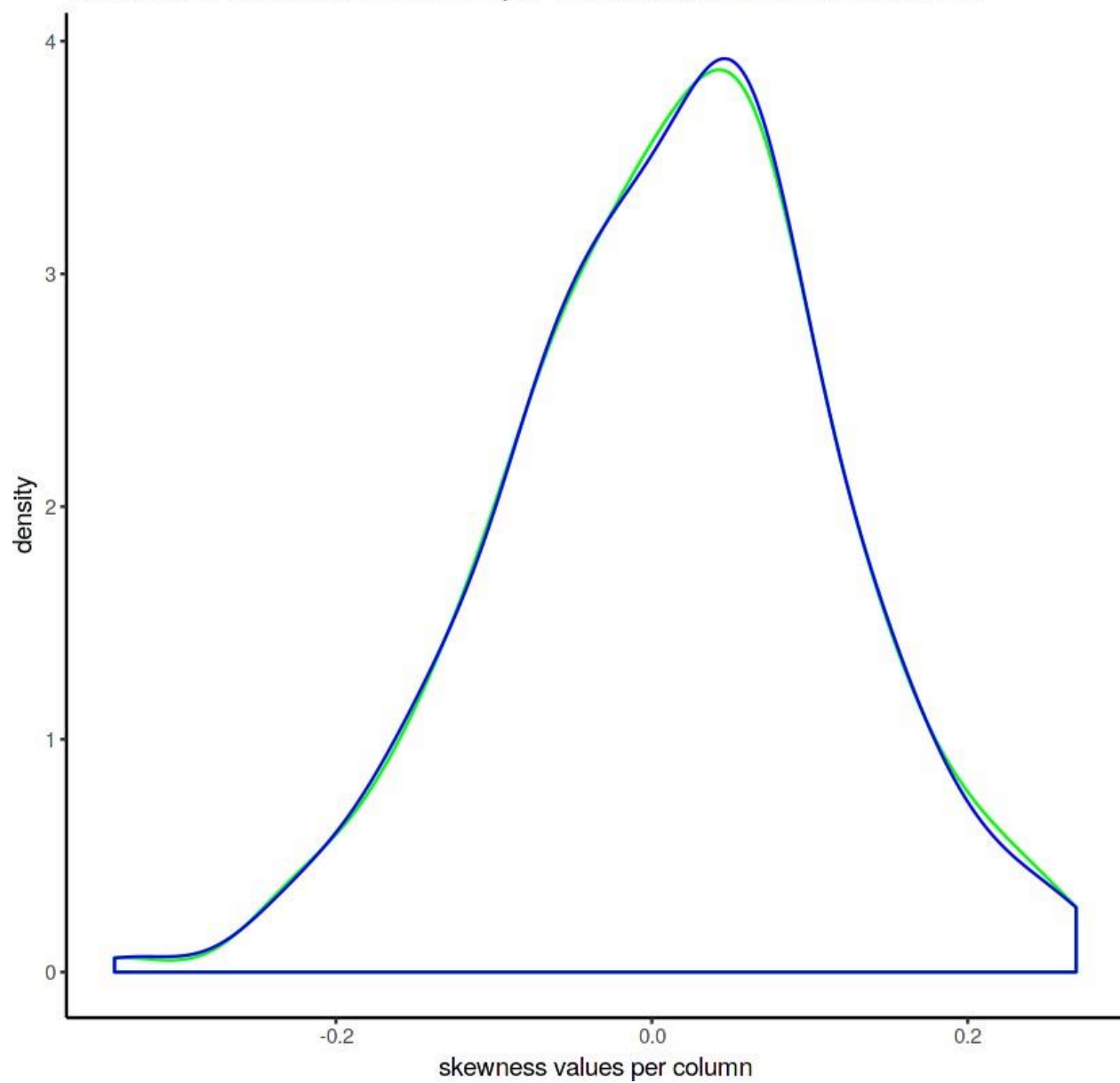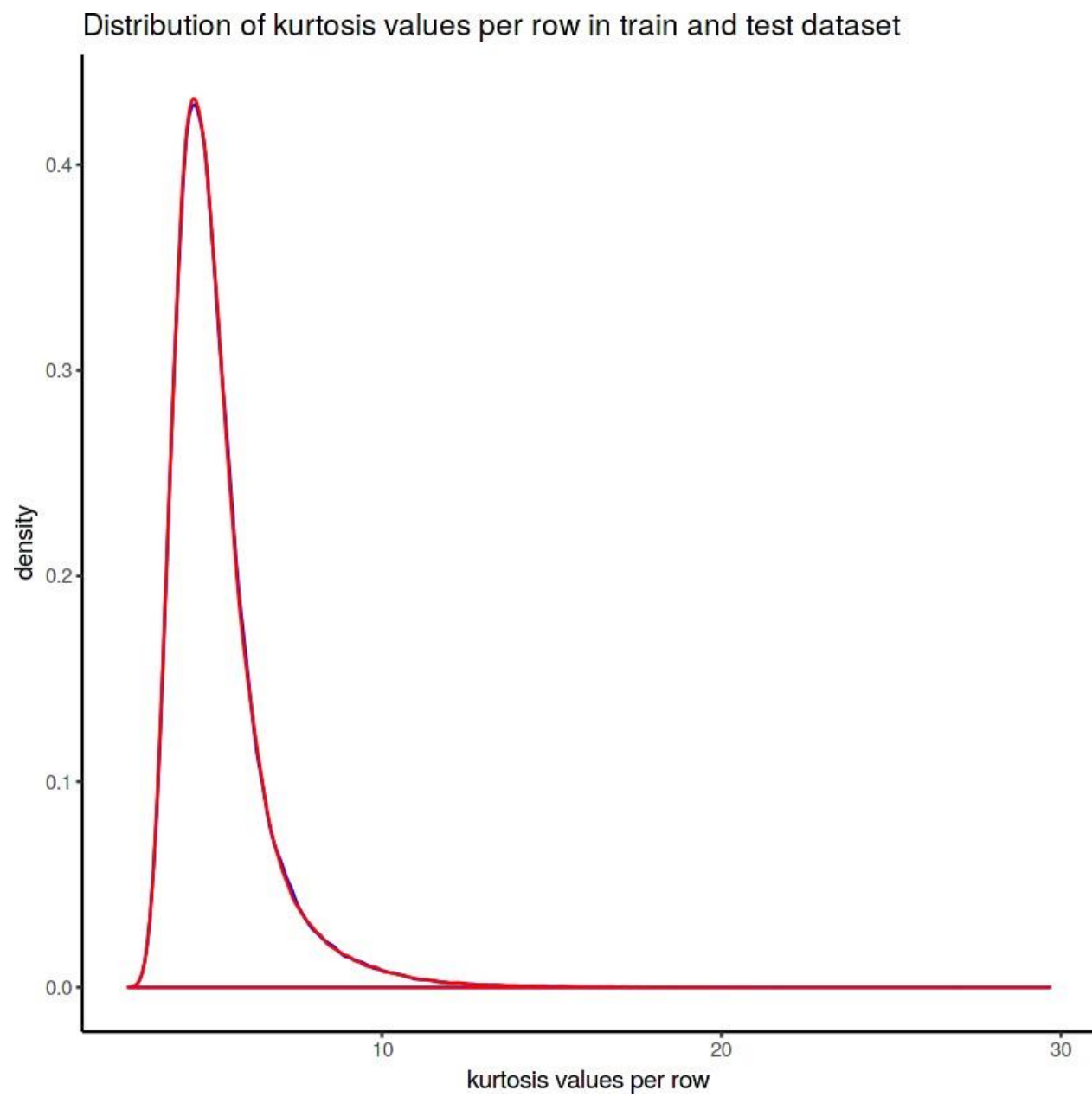
Distribution of kurtosis values per column in train and test dataset
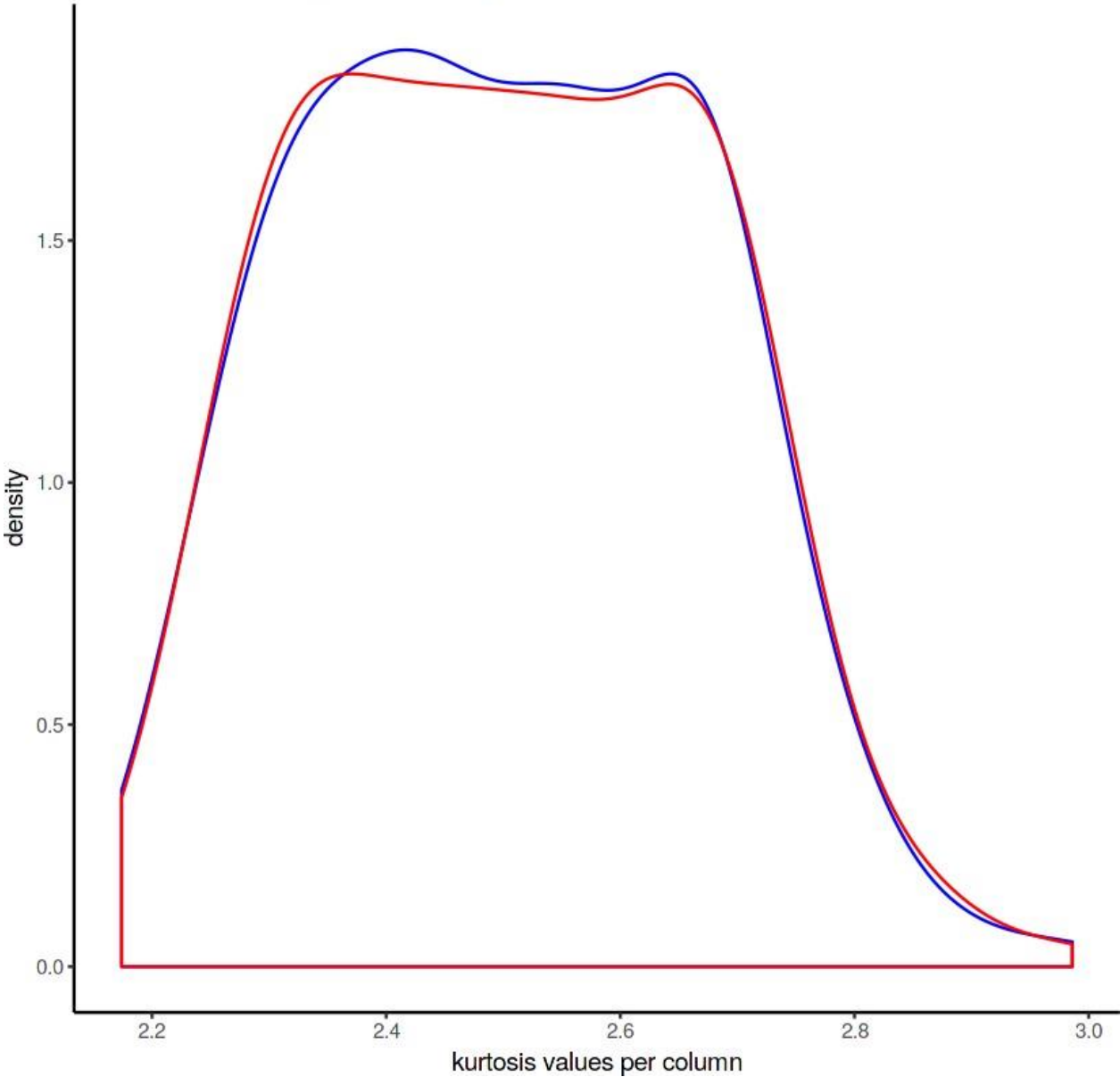
Appendix B – Co mp l et e Python a nd R Code
Python Code

Exploratory Data Analysis

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split,cross_val_predict,cross_val_
score
from sklearn.metrics import roc_auc_score,confusion_matrix,make_scorer,classifica
tion_report,roc_curve,auc
from sklearn.model_selection import StratifiedKFold
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import ClusterCentroids,NearMiss, RandomUnderSampler
import lightgbm as lgb
import eli5
from eli5.sklearn import PermutationImportance
from sklearn import tree
import graphviz
from pdpbox import pdp, get_dataset, info_plots
import scikitplot as skplt
from scikitplot.metrics import plot_confusion_matrix,plot_precision_recall_curve

from scipy.stats import randint as sp_randint
import warnings
warnings.filterwarnings('ignore')
random_state=42
np.random.seed(random_state)

#importing the train dataset
train_df=pd.read_csv('../input/train.csv')
train_df.head()

#Shape of the train dataset
train_df.shape

#Summary of the dataset
train_df.describe()
```

Target classes count

```
%%time
#target classes count
target_class=train_df['target'].value_counts()
print('Count of target classes :\n',target_class)
#Percentage of target classes count
per_target_class=train_df['target'].value_counts()/len(train_df)*100
print('percentage of count of target classes :\n',per_target_class)

#Countplot and violin plot for target classes
fig,ax=plt.subplots(1,2,figsize=(20,5))
sns.countplot(train_df.target.values,ax=ax[0],palette='husl')
sns.violinplot(x=train_df.target.values,y=train_df.index.values,ax=ax[1],
palette='husl')
sns.stripplot(x=train_df.target.values,y=train_df.index.values,jitter=True,
color=' black',linewidth=0.5,size=0.5,alpha=0.5,ax=ax[1],palette='husl')
ax[0].set_xlabel('Target')
ax[1].set_xlabel('Target')
ax[1].set_ylabel('Index')
```

Distribution of train attributes

```
%%time
def plot_train_attribute_distribution(t0,t1,label1,label2,train_attributes):
    i=0
    sns.set_style('whitegrid')

    fig=plt.figure()
    ax=plt.subplots(10,10,figsize=(22,18))

    for attribute in train_attributes:
        i+=1
        plt.subplot(10,10,i)
        sns.distplot(t0[attribute],hist=False,label=label1)
        sns.distplot(t1[attribute],hist=False,label=label2)
        plt.legend()
        plt.xlabel('Attribute',)
        sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
    plt.show()

Let us see first 100 train attributes

%%time
t0=train_df[train_df.target.values==0]
t1=train_df[train_df.target.values==1]
train_attributes=train_df.columns.values[2:102]
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)

Next 100 train attributes
train_attributes=train_df.columns.values[102:203]
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)
```

Distribution of test attributes

```
#importing the test dataset
test_df=pd.read_csv('../input/test.csv')
test_df.head()

#Shape of the test dataset
test_df.shape

def plot_test_attribute_distribution(test_attributes):
    i=0
    sns.set_style('whitegrid')

    fig=plt.figure()
    ax=plt.subplots(10,10,figsize=(22,18))

    for attribute in test_attributes:
        i+=1
        plt.subplot(10,10,i)
        sns.distplot(test_df[attribute],hist=False)
        plt.xlabel('Attribute',)
        sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
    plt.show()

#Let us see first 100 test attributes
test_attributes=test_df.columns.values[1:101]
plot_test_attribute_distribution(test_attributes)

 #Next 100 test attributes

test_attributes=test_df.columns.values[101:202]
plot_test_attribute_distribution(test_attributes)
```

Distribution of mean values in train and test dataset

```
%%time
#Distribution of mean values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
 #Distribution plot for mean values per column in train attributes
sns.distplot(train_df[train_attributes].mean(axis=0),color='blue',kde=True,bins=15
0,label='train')
 #Distribution plot for mean values per column in test attributes
sns.distplot(test_df[test_attributes].mean(axis=0),color='green',kde=True,bins=150
,label='test')
plt.title('Distribution of mean values per column in train and test dataset')
plt.legend()
plt.show()
```

```
#Distribution of mean values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for mean values per row in train attributes
sns.distplot(train_df[train_attributes].mean(axis=1),color='blue',kde=True,bins=15
0,label='train')
#Distribution plot for mean values per row in test attributes
sns.distplot(test_df[test_attributes].mean(axis=1),color='green',kde=True, bins=15
0, label='test')
plt.title('Distribution of mean values per row in train and test dataset')
plt.legend()
plt.show()
```

Distribution of standard deviation (std) in train and test dataset

```
%%time
#Distribution of std values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for std values per column in train attributes
sns.distplot(train_df[train_attributes].std(axis=0),color='red',kde=True,
bins=150,label='train')
#Distribution plot for std values per column in test attributes
sns.distplot(test_df[test_attributes].std(axis=0),color='blue',kde=True,bins=150,
label='test')
plt.title('Distribution of std values per column in train and test dataset')
plt.legend()
plt.show()

#Distribution of std values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for std values per row in train attributes
sns.distplot(train_df[train_attributes].std(axis=1),color='red',kde=True,bins=150
, label='train')
#Distribution plot for std values per row in test attributes
sns.distplot(test_df[test_attributes].std(axis=1),color='blue',kde=True, bins=150
,  label='test')
plt.title('Distribution of std values per row in train and test dataset')
plt.legend()
```

Distribution of skewness in train and test dataset

```
%%time
#Distribution of skew values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for skew values per column in train attributes
sns.distplot(train_df[train_attributes].skew(axis=0),color='green',kde=True,
```

```
 bins=150,label='train')
 #Distribution plot for skew values per column in test attributes
 sns.distplot(test_df[test_attributes].skew(axis=0),color='blue',kde=True,bins=150
 ,label='test')
 plt.title('Distribution of skewness values per column in train and test dataset')
 plt.legend()
 plt.show()

 #Distribution of skew values per row in train and test dataset
 plt.figure(figsize=(16,8))
 #Distribution plot for skew values per row in train attributes
 sns.distplot(train_df[train_attributes].skew(axis=1),color='green',kde=True,
 bins=150,label='train')
 #Distribution plot for skew values per row in test attributes
 sns.distplot(test_df[test_attributes].skew(axis=1),color='blue',kde=True,
 bins=150, label='test')
 plt.title('Distribution of skewness values per row in train and test dataset')
 plt.legend()
 plt.show()
```

Distribution of kurtosis values in train and test dataset

```
%%time
#Distribution of kurtosis values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for kurtosis values per column in train attributes
sns.distplot(train_df[train_attributes].kurtosis(axis=0),color='blue',kde=True,
bins=150,label='train')
#Distribution plot for kurtosis values per column in test attributes
sns.distplot(test_df[test_attributes].kurtosis(axis=0),color='red',kde=True,
bins=150,label='test')
plt.title('Distribution of kurtosis values per column in train and test dataset')
plt.legend()
plt.show()

#Distribution of kutosis values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for kurtosis values per row in train attributes
sns.distplot(train_df[train_attributes].kurtosis(axis=1),color='blue',kde=True,
bins=150,label='train')
#Distribution plot for kurtosis values per row in test attributes
sns.distplot(test_df[test_attributes].kurtosis(axis=1),color='red',kde=True,
bins=150, label='test')
plt.title('Distribution of kurtosis values per row in train and test dataset')
plt.legend()
plt.show()
```

Missing value analysis and Correlations

```
%%time
#Finding the missing values in train and test data
train_missing=train_df.isnull().sum().sum()
test_missing=test_df.isnull().sum().sum()
print('Missing values in train data :',train_missing)
print('Missing values in test data :',test_missing

%%time
#Correlations in train attributes
 train_attributes=train_df.columns.values[2:202]
 train_correlations=train_df[train_attributes].corr().abs().unstack().sort_values(k
ind='quicksort').reset_index()
train_correlations=train_correlations[train_correlations['level_0']!=train_correla
tions['level_1']]
 print(train_correlations.head(10))
 print(train_correlations.tail(10))

%%time
 #Correlations in test attributes
 test_attributes=test_df.columns.values[1:201]
 test_correlations=test_df[test_attributes].corr().abs().unstack().sort_values(kind
='quicksort').reset_index()
test_correlations=test_correlations[test_correlations['level_0']!=test_correlation
s['level_1']]
 print(test_correlations.head(10))
 print(test_correlations.tail(10))

 #Correlation plot

%%time
#Correlations in train data
 train_correlations=train_df[train_attributes].corr()
 train_correlations=train_correlations.values.flatten()
 train_correlations=train_correlations[train_correlations!=1]
 test_correlations=test_df[test_attributes].corr()
 #Correlations in test data
 test_correlations=test_correlations.values.flatten()
 test_correlations=test_correlations[test_correlations!=1]

 plt.figure(figsize=(20,5))
 #Distribution plot for correlations in train data
 sns.distplot(train_correlations, color="Red", label="train")
 #Distribution plot for correlations in test data
 sns.distplot(test_correlations, color="Blue", label="test")
 plt.xlabel("Correlation values found in train and test")
 plt.ylabel("Density")
 plt.title("Correlation distribution plot for train and test attributes")
 plt.legend()
```

Feature engineering

```python
#training data
X=train_df.drop(columns=['ID_code','target'],axis=1)
test=test_df.drop(columns=['ID_code'],axis=1)
y=train_df['target']

#Split the training data
X_train,X_valid,y_train,y_valid=train_test_split(X,y,random_state=42)

print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)

%%time
#Random forest classifier
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)
#fitting the model
rf_model.fit(X_train,y_train)

#Permutation importance

%%time
from eli5.sklearn import PermutationImportance
perm_imp=PermutationImportance(rf_model,random_state=42)
#fitting the model
perm_imp.fit(X_valid,y_valid)

%%time
#Important features
eli5.show_weights(perm_imp,feature_names=X_valid.columns.tolist(),top=200)

#partial dependence plots

%%time
#Create the data we will plot 'var_81'
features=[v for v in X_valid.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature=
'var_81')
#plot feature "var_81"
pdp.pdp_plot(pdp_data,'var_81')
plt.show()

%%time
#Create the data we will plot
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature=
' var_109')
#plot feature "var_109"
pdp.pdp_plot(pdp_data,'var_109')
plt.show()
```

Handling of imbalanced data

```python
#Training data
X=train_df.drop(['ID_code','target'],axis=1)
Y=train_df['target']
#StratifiedKFold cross validator
cv=StratifiedKFold(n_splits=5,random_state=42,shuffle=True)
for train_index,valid_index in cv.split(X,Y):
    X_train, X_valid=X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid=Y.iloc[train_index], Y.iloc[valid_index]

print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)

%%time
#Logistic regression model
lr_model=LogisticRegression(random_state=42)
#fitting the lr model
lr_model.fit(X_train,y_train)

#Accuracy of the model
lr_score=lr_model.score(X_train,y_train)
print('Accuracy of the lr_model :',lr_score)

%%time
#Cross validation prediction
cv_predict=cross_val_predict(lr_model,X_valid,y_valid,cv=5)
#Cross validation score
cv_score=cross_val_score(lr_model,X_valid,y_valid,cv=5)
print('cross_val_score :',np.average(cv_score))

#Confusion matrix
cm=confusion_matrix(y_valid,cv_predict)
#Plot the confusion matrix
plot_confusion_matrix(y_valid,cv_predict,normalize=False,figsize=(15,8))

#ROC_AUC score
roc_score=roc_auc_score(y_valid,cv_predict)
print('ROC score :',roc_score)

#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_valid,cv_predict)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
```

```
plt.show()
print('AUC:',roc_auc)

#Classification report
scores=classification_report(y_valid,cv_predict)
print(scores)

%%time
#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
lr_pred=lr_model.predict(X_test)
print(lr_pred)

#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_valid,y_valid)

#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)

#Accuracy of the model
smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)

#Cross validation prediction
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score :',np.average(cv_score))

#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
#Plot the confusion matrix
plot_confusion_matrix(y_smote_v,cv_pred,normalize=False,figsize=(15,8))

#ROC_AUC score
roc_score=roc_auc_score(y_smote_v,cv_pred)
print('ROC score :',roc_score)

#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_smote_v,cv_pred)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
```

```python
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)

#Classification report
scores=classification_report(y_smote_v,cv_pred)
print(scores)

%%time
#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
print(smote_pred)
```

LightGBM

```python
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)

#Selecting best hyperparameters by tuning of different parameters

params={'boosting_type': 'gbdt',
        'max_depth' : -1, #no limit for max_depth if <0
        'objective': 'binary',
        'boost_from_average':False,
        'nthread': 8,
        'metric':'auc',
        'num_leaves': 100,
        'learning_rate': 0.03,
        'max_bin': 950,       #default 255
        'subsample_for_bin': 200,
        'subsample': 1,
        'subsample_freq': 1,
        'colsample_bytree': 0.8,
        'reg_alpha': 1.2, #L1 regularization(>0)
        'reg_lambda': 1.2,#L2 regularization(>0)
        'min_split_gain': 0.5, #>0
        'min_child_weight': 1,
        'min_child_samples': 5,
        'is_unbalance':True,
        }

num_rounds=3000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],
verbose_eval=100,early_stopping_rounds = 1000)

X_test=test_df.drop(['ID_code'],axis=1)
#predict the model
#probability predictions
lgbm_predict_prob=lgbm.predict(X_test,random_state=42,num_iteration=lgbm.best_iter
ation)
```

```python
#Convert to binary output 1 or 0
lgbm_predict=np.where(lgbm_predict_prob>=0.5,1,0)
print(lgbm_predict_prob)
print(lgbm_predict)

#plot the important features
lgb.plot_importance(lgbm,max_num_features=150,importance_type="split",figsize=(20,
50))

#final submission
sub_df=pd.DataFrame({'ID_code':test_df['ID_code'].values})
sub_df['lgbm_predict_prob']=lgbm_predict_prob
sub_df['lgbm_predict']=lgbm_predict
sub_df['smote_predict']=smote_pred
sub_df.to_csv('submission.csv',index=False)
sub_df.head()
```

R Code

Exploratory Data Analysis

```
#Load the libraries
library(tidyverse)
library(moments)
library(DataExplorer)
library(caret)
library(Matrix)
library(mlbench)
library(caTools)
library(randomForest)
library(glmnet)
library(mlr)
library(unbalanced)
library(vita)
library(rBayesianOptimization)
library(lightgbm)
library(boot)
library(pROC)
library(DMwR)
library(ROSE)
library(yardstick)

#loading the train data
train_df<-read.csv('../input/train.csv')
head(train_df)

#Dimension of train data
dim(train_df)

#Summary of the dataset
str(train_df)
#convert to factor
train_df$target<-as.factor(train_df$target)
```

Target classes count in train data

```
require(gridExtra)
#Count of target classes
table(train_df$target)
#Percenatge counts of target classes
table(train_df$target)/length(train_df$target)*100
#Bar plot for count of target classes
plot1<-ggplot(train_df,aes(target))+theme_bw()+geom_bar(stat='count',fill='lig
htgreen')
#Violin with jitter plots for target classes
plot2<-ggplot(train_df,aes(x=target,y=1:nrow(train_df)))+theme_bw()+geom_violi
n(fill='lightblue')+
```

```
facet_grid(train_df$target)+geom_jitter(width=0.02)+labs(y='Index')
grid.arrange(plot1,plot2, ncol=2)
```

Distribution of train attributes

```
#Distribution of train attributes from 3 to 102
for (var in names(train_df)[c(3:102)]){
    target<-train_df$target
    plot<-ggplot(train_df, aes(x=train_df[[var]],fill=target)) +
    geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
    print(plot)
}

#Distribution of train attributes from 103 to 202
for (var in names(train_df)[c(103:202)]){
    target<-train_df$target
    plot<-ggplot(train_df, aes(x=train_df[[var]], fill=target)) +
    geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
    print(plot)
}
```

Distribution of test attributes

```
#loading test data
test_df<-read.csv('../input/test.csv')
head(test_df)

#Dimension of test dataset
dim(test_df)

#Distribution of test attributes from 2 to 101
plot_density(test_df[,c(2:101)], ggtheme = theme_classic(),
geom_density_args =  list(color='blue'))

#Distribution of test attributes from 102 to 201
plot_density(test_df[,c(102:201)], ggtheme = theme_classic(),
geom_density_args = list(color='blue'))
```

Distribution of mean values in train and test dataset

```
#Applying the function to find mean values per row in train and test data.
train_mean<-apply(train_df[,-c(1,2)],MARGIN=1,FUN=mean)
test_mean<-apply(test_df[,-c(1)],MARGIN=1,FUN=mean)
ggplot()+
#Distribution of mean values per row in train data
geom_density(data=train_df[,-c(1,2)],aes(x=train_mean),kernel='gaussian',
show. legend=TRUE,color='blue')+theme_classic()+
```

```
#Distribution of mean values per row in test data
geom_density(data=test_df[,-c(1)],aes(x=test_mean),kernel='gaussian',
show.legend=TRUE,color='green')+
labs(x='mean values per row',title="Distribution of mean values per row in
train and test dataset")

#Applying the function to find mean values per column in train and test data.
train_mean<-apply(train_df[,-c(1,2)],MARGIN=2,FUN=mean)
test_mean<-apply(test_df[,-c(1)],MARGIN=2,FUN=mean)
ggplot()+
#Distribution of mean values per column in train data
geom_density(aes(x=train_mean),kernel='gaussian',show.legend=TRUE,
color='blue')+theme_classic()+
#Distribution of mean values per column in test data
geom_density(aes(x=test_mean),kernel='gaussian',show.legend=TRUE,
color='green')+
labs(x='mean values per column',title="Distribution of mean values per row
in  train and test dataset")
```

Distribution of standard deviation in train and test dataset

```
#Applying the function to find standard deviation values per row in train
and  test data.
train_sd<-apply(train_df[,-c(1,2)],MARGIN=1,FUN=sd)
test_sd<-apply(test_df[,-c(1)],MARGIN=1,FUN=sd)
ggplot()+
#Distribution of sd values per row in train data
geom_density(data=train_df[,-c(1,2)],aes(x=train_sd),kernel='gaussian',
show.le gend=TRUE,color='red')+theme_classic()+
#Distribution of mean values per row in test data
geom_density(data=test_df[,-c(1)],aes(x=test_sd),kernel='gaussian',
show.legend=TRUE,color='blue')+
labs(x='sd values per row',title="Distribution of sd values per row in
train and test dataset")

#Applying the function to find sd values per column in train and test data.
train_sd<-apply(train_df[,-c(1,2)],MARGIN=2,FUN=sd)
test_sd<-apply(test_df[,-c(1)],MARGIN=2,FUN=sd)
ggplot()+
#Distribution of sd values per column in train data
geom_density(aes(x=train_sd),kernel='gaussian',show.legend=TRUE,color='red')+
theme_classic()+
#Distribution of sd values per column in test data
geom_density(aes(x=test_sd),kernel='gaussian',show.legend=TRUE,color='blue')+
labs(x='sd values per column',title="Distribution of std values per column
in train and test dataset")
```

Distribution of skewness values in train and test dataset

```
#Applying the function to find skewness values per row in train and test data.
```

```
train_skew<-apply(train_df[,-c(1,2)],MARGIN=1,FUN=skewness)
test_skew<-apply(test_df[,-c(1)],MARGIN=1,FUN=skewness)
ggplot()+
#Distribution of skewness values per row in train data
geom_density(aes(x=train_skew),kernel='gaussian',show.legend=TRUE,
color='green')+theme_classic()+
#Distribution of skewness values per column in test data
geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,
color='blue') +labs(x='skewness values per row',title="Distribution of
skewness values per row in train and test dataset")

#Applying the function to find skewness values per column in train and
test data.
train_skew<-apply(train_df[,-c(1,2)],MARGIN=2,FUN=skewness)
test_skew<-apply(test_df[,-c(1)],MARGIN=2,FUN=skewness)
ggplot()+
#Distribution of skewness values per column in train data
geom_density(aes(x=train_skew),kernel='gaussian',show.legend=TRUE,
color='green')+theme_classic()+
#Distribution of skewness values per column in test data
geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,
color='blue')+labs(x='skewness values per column',title="Distribution of
skewness values per column in train and test dataset")
```

Distribution of kurtosis values in train and test dataset

```
#Applying the function to find kurtosis values per row in train and test data.
train_kurtosis<-apply(train_df[,-c(1,2)],MARGIN=1,FUN=kurtosis)
test_kurtosis<-apply(test_df[,-c(1)],MARGIN=1,FUN=kurtosis)
ggplot()+
#Distribution of sd values per column in train data
geom_density(aes(x=train_kurtosis),kernel='gaussian',show.legend=TRUE,
color='b lue')+theme_classic()+
#Distribution of sd values per column in test data
geom_density(aes(x=test_kurtosis),kernel='gaussian',show.legend=TRUE,
color='red')+labs(x='kurtosis values per row',title="Distribution of
kurtosis values per row in train and test dataset")

#Applying the function to find kurtosis values per column in train and
test data.
train_kurtosis<-apply(train_df[,-c(1,2)],MARGIN=2,FUN=kurtosis)
test_kurtosis<-apply(test_df[,-c(1)],MARGIN=2,FUN=kurtosis)
ggplot()+
#Distribution of sd values per column in train data
geom_density(aes(x=train_kurtosis),kernel='gaussian',show.legend=TRUE,
color='blue')+theme_classic()+
#Distribution of sd values per column in test data
geom_density(aes(x=test_kurtosis),kernel='gaussian',show.legend=TRUE,
color='red')+
labs(x='kurtosis values per column',title="Distribution of kurtosis values
per  column in train and test dataset")
```

Missing value analysis and Correlations

```
#Finding the missing values in train data
missing_val<-data.frame(missing_val=apply(train_df,2,
function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
missing_val

#Finding the missing values in test data
missing_val<-data.frame(missing_val=apply(test_df,2,
function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
missing_val

#Correlations in train data
train_df$target<-as.numeric(train_df$target)
train_correlations<-cor(train_df[,c(2:202)])
train_correlations

#Correlations in test data
test_correlations<-cor(test_df[,c(2:201)])
test_correlations
```

Feature Engineering

```
#Split the training data
train_index<-sample(1:nrow(train_df),0.75*nrow(train_df))
train_data<-train_df[train_index,]
valid_data<-train_df[-train_index,]
dim(train_data)
dim(valid_data)

#Training the Random forest classifier
set.seed(2732)
train_data$target<-as.factor(train_data$target)
mtry<-floor(sqrt(200))
tuneGrid<-expand.grid(.mtry=mtry)
rf<-randomForest(target~.,train_data[,-c(1)],mtry=mtry,ntree=10,
importance=TRUE)

#Variable importance
VarImp<-importance(rf,type=2)
VarImp

#Partial dependence plot

#We will plot "var_81"
partialPlot(rf,valid_data[,-c(1,2)],valid_data$var_81,xlab='var_81

#We will plot "var_12"
partialPlot(rf,valid_data[,-c(1,2)],valid_data$var_12,xlab='var_12')
```

Handling of imbalanced data

```
#Split the data using CreateDataPartition
set.seed(689)
train.index<-createDataPartition(train_df$target,p=0.8,list=FALSE)
train.data<-train_df[train.index,]
valid.data<-train_df[-train.index,]
dim(train.data)
dim(valid.data)

#training dataset
set.seed(682)
X_t<-as.matrix(train.data[-c(1,2)])
y_t<-as.matrix(train.data$target)
#validation dataset
X_v<-as.matrix(valid.data[-c(1,2)])
y_v<-as.matrix(valid.data$target)
#test data
test<-as.matrix(test_df[,-c(1)])

#Logistic regression model
set.seed(667)
lr_model <-glmnet(X_t, y_t, family = "binomial")
summary(lr_model)

#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
cv_lr

#Minimum lambda
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)

#Model performance on validation dataset
set.seed(5363)
cv_predict.lr<-predict(cv_lr,X_v,s = "lambda.min", type = "class")
cv_predict.lr
```

```
#Confusion matrix
set.seed(689)
#actual target variable
target<-valid.data$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)
```

```
#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
roc(data=valid.data[,-c(1,2)],response=target,predictor=cv_predict.lr,
auc=TRUE,plot=TRUE)

#predict the model
set.seed(763)
lr_pred<-predict(lr_model,test,type='class')
lr_pred

#Random Oversampling Examples(ROSE)
set.seed(699)
#train.data$target<-as.factor(train.data$target)
train.rose <- ROSE(target~., data =train.data[,-c(1)],seed=32)$data
table(train.rose$target)
valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=32)$data
table(valid.rose$target)

#Logistic regression model
set.seed(462)
lr_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family =
"binomial")
summary(lr_rose)

#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),
family = "binomial", type.measure = "class")
cv_rose

#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)

#Model performance on validation dataset
set.seed(442)
cv_predict.rose<-predict(cv_rose,as.matrix(valid.rose),s = "lambda.min",
type = "class")
cv_predict.rose

#Confusion matrix
set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
```

```r
cv_predict.rose<-as.factor(cv_predict.rose)
confusionMatrix(data=cv_predict.rose,reference=target)

#ROC_AUC score and curve
set.seed(843)
cv_predict.rose<-as.numeric(cv_predict.rose)
roc(data=valid.rose[,-c(1,2)],response=target,predictor=cv_predict.rose,
auc=TRUE,plot=TRUE)

#predict the model
set.seed(6543)
rose_pred<-predict(lr_rose,test,type='class')
rose_pred

#Convert data frame to matrix
set.seed(5432)
X_train<-as.matrix(train.data[,-c(1,2)])
y_train<-as.matrix(train.data$target)
X_valid<-as.matrix(valid.data[,-c(1,2)])
y_valid<-as.matrix(valid.data$target)
test_data<-as.matrix(test_df[,-c(1)])

#training data
lgb.train <- lgb.Dataset(data=X_train, label=y_train)
#Validation data
lgb.valid <- lgb.Dataset(data=X_valid,label=y_valid)

set.seed(653)
lgb.grid = list(objective = "binary",
                metric = "auc",
                min_sum_hessian_in_leaf = 1,
                feature_fraction = 0.7,
                bagging_fraction = 0.7,
                bagging_freq = 5,
                learning_rate=0.1,
                num_leaves=100,
                num_threads=8,
                min_data = 100,
                max_bin = 200,
                lambda_l1 = 8,
                lambda_l2 = 1.3,
                min_data_in_bin=150,
                min_gain_to_split = 20,
                min_data_in_leaf = 40,
                is_unbalance = TRUE)

set.seed(7663)
lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =3000,
eval_freq =100,valids=list(val1=lgb.train,val2=lgb.valid),
early_stopping_rounds = 1000)
```

```
#lgbm model performance on test data
set.seed(6532)
lgbm_pred_prob <- predict(lgbm.model,test_data)
print(lgbm_pred_prob)
#Convert to binary output (1 and 0) with threshold 0.5
lgbm_pred<-ifelse(lgbm_pred_prob>0.5,1,0)
print(lgbm_pred)

set.seed(6521)
tree_imp <- lgb.importance(lgbm.model, percentage = TRUE)
lgb.plot.importance(tree_imp, top_n = 150, measure = "Gain")

sub_df<-data.frame(ID_code=test_df$ID_code,lgb_predict_prob=lgbm_pred_prob,
lgb  _predict=lgbm_pred,smote_predict=smote_pred)
write.csv(sub_df,'submission.CSV',row.names=F)
head(sub_df)
```

References

[1]. https://www.kaggle.com

[2]. http://rprogramming.net/

[3]. https://medium.com/

[4]. **Practical Machine learning with Python** book by by Dipanjan Sarkar , Raghav Bali and Tushar Sharma

[5]. https://stackoverflow.com/

[6]. https://www.analyticsvidhya.com/blog